Internship report

Cycle: IT2

Automatic detection and correction of earthquakes and slow slip events in GNSS station time series

Clément Drouadaine

16 September 2014

[✓ *Non confidential*] □ *Confidential IGN* □ *Confidential Industry* □ *Until...*

Jury

President of jury:

Alain DUPÉRET

Host Company:

Land Information New Zealand (LINZ)

Internship training supervisor:

Chris CROOK, LINZ

Xavier COLLILIEUX, IGN

Educational person in charge of the engineering course:

Serge BOTTON

School's professional training counsellor:

Patricia PARISI

Internship *from 30/05/2014 to 15/08/2014*

Web diffusion: [✓ *Internet*] [✓ *ENSG Intranet*]

Document situation:

internship report presented in the end of the 2nd year of the engineer cycle

Number of pages: *48 including 15 of annexes*

Host system: *Word*

MODIFICATIONS

EDITION	*REVISION*	*DATE*	*CHANGES*
1	*0*	*26/08/2002*	*Creation*
1	*1*	*04/09/2013*	*Logo*

«Albert grunted. “Do you know what happens to lads who ask too many questions?” Mort thought for a moment. “No,” he said eventually, “what?” There was silence. Then Albert straightened up and said, “Damned if I know. Probably they get answers, and serve ’em right.”»
from Mort, the fourth Discworld novel, by Sir Terry Pratchett.

Acknowledgements

I want to thank my internship supervisor in LINZ, Chris Crook, for all the help and support he provided me all along the internship, and his availability whenever I needed him.

I would also like to thank my French supervisor, Xavier Collilieux, for his remarks, suggestions, the documentation he provided me, and his general guidance.

Thanks to the whole of the topodetic team for their welcome, and to Sam Fisher for her cakes.

And finally, thanks to Zuheir Altamimi, for having found me this internship within 24 hours after I asked him.

Résumé

La Nouvelle-Zélande, situé à la frontière de deux plaques tectoniques, est sujet à une forte activité sismique. Le réseau GNSS permanent en Nouvelle-Zélande, qui surveille les mouvements du sol, en est un témoin privilégié, car il se déplace avec lui. Tous les tremblements de terre et les séismes lents (changement temporaire et géographiquement localisée de la vitesse de déplacement de la plaque) rendent très difficile la modélisation de la position d'une station. Ma tâche, et l'objectif de mon stage, étaient d'automatiser ce traitement.

J'ai commencé par l'automatisation du nettoyage des séries temporelles brutes, à savoir l'élimination des valeurs aberrantes et du bruit. Ensuite, j'ai créé un algorithme, librement inspiré d'un créé par la NASA pour détecter les séismes, pour détecter tous les événements sismiques, à la fois séismes et glissements lents. Compte tenu de l'efficacité, la précision et le gain de temps de apportés par ce nouveau processus, il est appelé à remplacer l'ancienne détection visuelle des événements, et à devenir la norme à LINZ.

Mots-clefs: automatisation, séries temporelles, détection de sauts, séismes, glissements lents.

Abstract

New Zealand, being astride the boundary between two tectonic plates, experiences a significant amount of seismic activity. The New Zealand permanent GNSS network, which monitors the earth's motion there, is a direct witness of this activity, since it moves with it. All the earthquakes and slow earthquakes (i.e. temporary and geographically localised change in the earth velocity) make the modelling of a station's position very difficult.

My task, and the goal of this internship, was to automate this treatment. I started by automating the cleaning of the raw time series, i.e. removing their noise and outliers. Then, I created an algorithm, based upon one made by NASA, to detect tectonic events in the time series, both earthquakes and slow slip events. Given the efficiency, the precision, and the gain of time of this new process, it should now replace the old visual detection of events, and become standard.

Keywords: automation, time series, jump detection, offsets, earthquakes, slow slips.

Table of contents

List of figures

List of annexes

Glossary and useful acronyms

ECEF: earth-centred earth-fixed, also known as cartesian, a type of geodetic coordinates

MAD: median absolute deviation, statistical tool based on the deviation of the elements to the median of the sample

Slow slip, slow earthquake: discontinuous, earthquake-like event that releases energy over a period of hours to months, rather than the seconds to minutes characteristic of a typical earthquake

WRMS: weighted root mean square, also known as the quadratic mean. A statistical measure of the magnitude of a varying quantity

INTRODUCTION

New Zealand, being astride the boundary between the Pacific and Australian tectonic plates, is a very geologically active, and subject to frequent earthquakes. The GNSS network is a direct witness of that: some parts of the country are moving at a speed of around 40mm/year, and slow slip earthquakes can cause stations to be displaced of up to several metres. On top of that, there are also equipment changes that may too cause a discontinuity in the observations. The corresponding GNSS time series are thus very irregular, and finding models to fit these observations is not an easy task.

At LINZ, there already exists software for fitting a model but it only automates fitting a simple model of a constant station velocity model and cyclic (seasonal) effect, and does not account for jumps or slope changes. It is then up to the operator to manually signal events by clicking on the areas of the graphics where he sees problems. This process must then be repeated for each station. It is obviously quite imprecise and very time-consuming.

The goal of this internship is to create a routine to automatically detect seismic events in the GNSS time series. These events can either be proper earthquakes, which are simply discontinuities in the time series, or slow slips, i.e. earthquakes that happen over a significant period of time (a few days to a few months), causing an important change in the model velocity for their duration.

FIRST PART: USING BUILT-IN FUNCTIONS

Cleaning data and removing outliers

What does our data look like?

The data we will be working with is composed of just one text file per station, for all of the 39 stations of the PositioNZ network. These files are of course based on the same model, and are composed of one line of header followed by one line per day of observation.
Said lines are composed of:
- the code for the station;
- the date of the observation, in ISO8601 format (for example, 2000-01-05T12:00:00 for an observation on the 5 January 2000);
- the position measured, in ECEF coordinates, i.e. XYZ from the centre of the Earth;
- a one-character flag, which I did not use.

Being in ECEF, these coordinates needed to be converted into an East-North-Up format for the beginning of the treatment: this way, most the noise would go to the Up coordinate instead of being distributed across all of the three XYZ dimensions, which would be a better representation of the real phenomenon, due to the physics of the GNSS. It will allow better representation of the coordinates and accuracies without requiring full coordinate covariance matrices.

See annex 1 for an example of observation file.

Which programming language?

At the beginning of this internship, I could choose to work in any programming language I liked, since I would start my software from scratch. However, one of the favourite languages of my supervisor was Python, and it was in this language that he had coded the previous software for a visual detection of earthquakes, which I might have to take parts from.
Moreover, even if I was not used to Python, I had had a short introduction to it at school, and I knew, and liked, its spirit and principles. It is a very high level and intuitive language, and was thus indicated for quick self-learning. Additionally, the Python numpy, scipy, and matplotlib modules provide a toolkit for efficient manipulation and plotting of time series data.

Why cleaning the data?

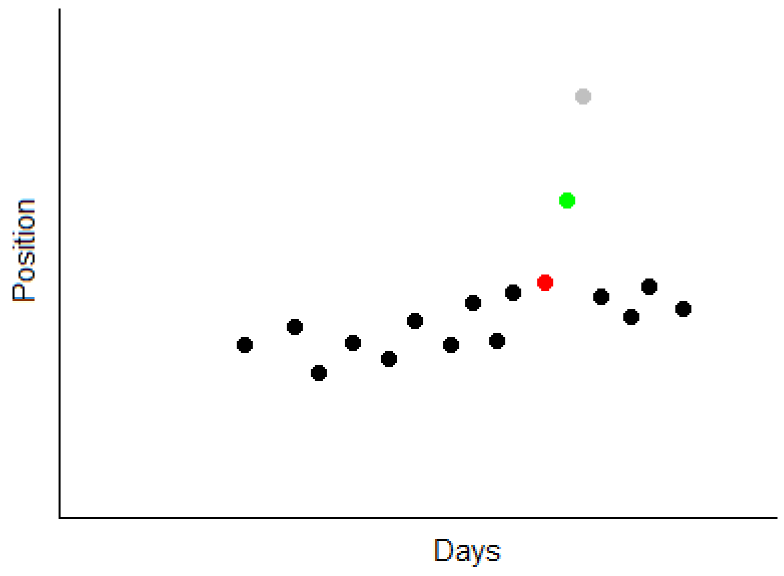
Before processing the time series, we need to clean them. The most obvious thing to do is to remove outliers that could otherwise affect our calculations. "An outlying observation, or "outlier", is one that appears to deviate markedly from other members of the sample in which it occurs." [1] Indeed, the measurement of the quality of the result often relies on squared differences between the model and the observations, and outliers are very likely to introduce a strong bias in such a measurement.
Moreover, there is a strong presence of noise in our data (roughly ±3 mm). This can lead to difficulties to distinguish the signal from the noise, especially in our context of precisely detecting changes in the values of said data. This is why we will also have to smooth the data, to reduce the noise and to allow for a better detection of real events.

Removing outliers

As well as computing the corrections to apply to each day, process which will be detailed later, I looked for outliers, in order to remove them [2, 3], and thus to prevent them from having any influence on the corrections I apply. To this end, I tried different outlier-detection methods, three of which will be described here. All have in common that they work day-by-day, testing one observation after the other, and they also share the dependency on a subset of time series, based on a window of 5 to 8 days around the observation we are examining.

1st Method. Using average, the easiest method to implement and one of the most effective. First, I compute the average of all neighbouring points, without taking into account the observation itself. If the difference between the observation and the average is not too important (the threshold I used was 2 mm), I just proceed to the next observation. Otherwise, the observation is marked as a possible outlier; it is not straight away marked as an outlier, since there is always the possibility that there is another outlier nearby, which would affect the average.

Fig 1. An example of incorrect outlier detection: the red dot is detected as outlier, because of the two extreme values just after it.



To deal with the problem of close outliers, once a point A is suspected of being an outlier, we look at the points close to it, test each point, and look at the point with the biggest deviation from the mean. If it is the same point as before, it is definitely an outlier. Else, it marked as a possible outlier, and the process is repeated. Once the outlier is found, it is outright deleted from the data. Then, we get back to our point A, and we start again. In the example given in figure 1, we will first detect the red point, then the green one, then the grey one, which will be deleted. We come back to the red, that is once again detected, then the grey, that is deleted, we come back once more to the red, which will then not be detected since it is now close to the average.

The threshold to decide of which points are to be deleted, that is to say the M such as an observation is deleted if |*observation – average*| > *M*, is empirically determined to get visually satisfactory data, without removing too much points, and is fixed. This method, as well as the other methods described here, would doubtlessly work better with dynamic thresholds, based on standard deviations or other mathematical indicators. I did not take the time to implement these things, and I will explain why in a moment.

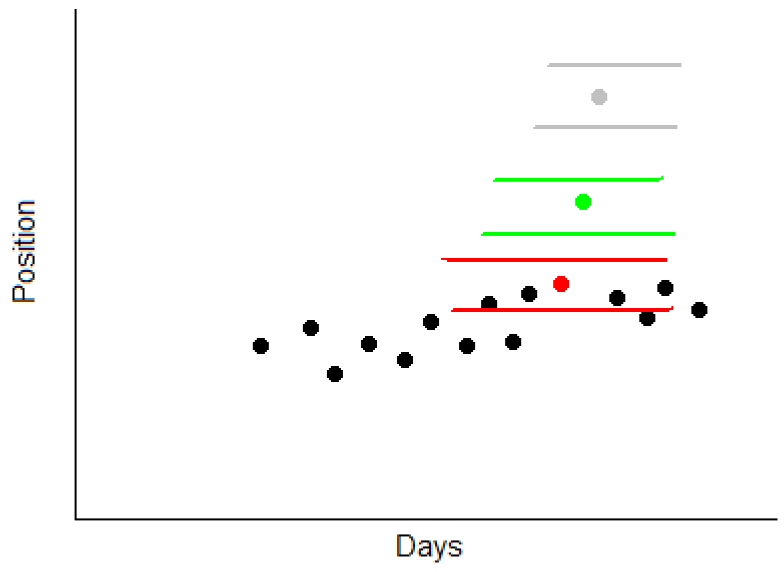
2nd method. Using the Median Absolute Deviation (MAD) instead of the average [4]. The MAD is basically just another statistical tool, more robust than the average, and thus less sensitive to the very outliers we are trying to detect. Its formula is

(1) $MAD = bM(|x_i – M(x_j) |)$

where x_i and x_j are the original observations, M is the median function, and b = *1/Q(0.75)*, where *Q(0.75)* is the 0.75 quartile of the distribution. With a normal distribution, b = 1.4826. Since my distribution is not exactly normal, and I did not succeed in getting the quartile right, this method obviously gave bad results.

3rd method. Using the number of neighbours. For each point, we count the number of observations which are within a range of M around it. If this number is less than a certain threshold, the point is rejected. Once again, M and the threshold are empirically chosen by the operator. The efficacy of this method depends greatly on the level of noise of the data.

Fig 2. Using neighbouring points. The red dot has 4 neighbours, the grey and green ones have none.



Miscellaneous:

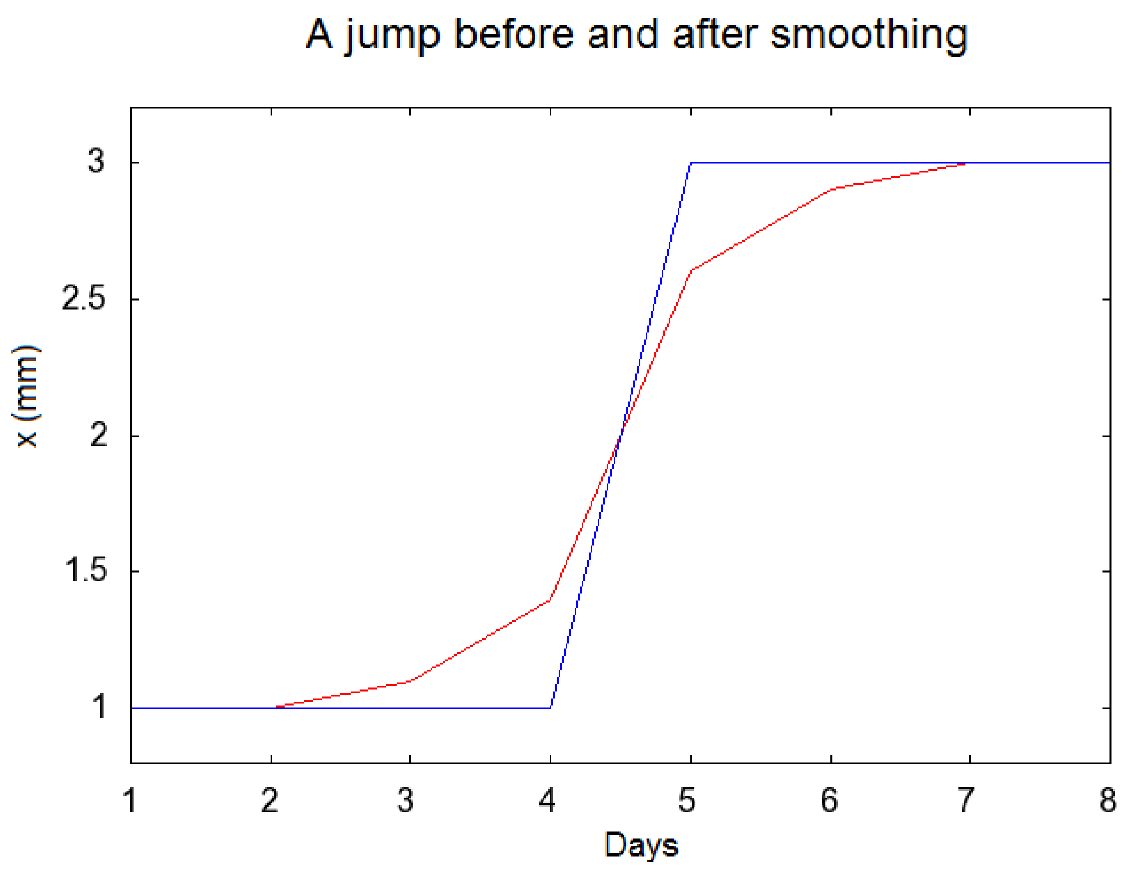
- Each time a point is rejected as an outlier, we go back in the treatment to compute again all the points which could have been affected by it.
- We look at outliers coordinate by coordinate. If a point is detected as an outlier on one coordinate, its whole data is erased. However, the Up coordinate being a lot noisier than the others, it is often not taken into account, or with really bigger thresholds.
- A few problems arise of this coordinate-by-coordinate process, when, in the phenomenon we observe, coordinates are linked altogether. For example, using method 1 with a threshold of rejection fixed at 5, a point whose deviations from the mean in East North Up are (0, 5.01, 0) will be rejected when another with (-4.9, 4.8, 3.7) will be kept, even if it appears “more wrong” than the first one. We might have had better results if we had tried to do some sort of average of the deviations taking account of the global level of noise of the coordinates.
- With a window of size 7, we will look at all the observations within one week of the current one. If there is missing data, we do not try to get further away data to compensate. If there is not enough data around our point (typically less than half of the size of the window), we will not process the point, and go on to the next observation. This can pose problems for outliers in the middle of a gap in the observations, which will stay where they are; or if we have several close outliers: as they get removed, the number of points in the window decreases, and this can lead to outlier being not removed, simply because there were too many outliers in the same area.

Smoothing the data

The underlying idea behind the smoothing of the data is to process observation by observation and to work with mini-time series centred on the observation. We replace the observation by the mean of this sliding window, or its median, or another statistical tool, such as the MAD described previously. Should there be a missing observation, we go without it rather than adding data further away to compensate, as it would not make much sense, and the gaps in data can be quite big.

The window length I chose is around 5 days on both sides of the observations, in order to have enough data for it to be meaningful, not too sensitive to noise, and to actually change the observation, while at the same time being not too big. With a too big window, we would smooth the edges of jumps, and make their detection harder, we could have several jumps in the same window, and taking into account observations made more than one week away from the observation just does not make a lot of sense anyway.

Fig 3. A jump before (blue) and after smoothing (red). Smoothing makes the edges of the jump less neat.



The first thing I tried was to work with the average of the mini-time series. As suggested, I would compute the average of the window, and, from that, get a correction to apply to the observation in order to make it equal to the average. That would produce not-so-bad results for most of the observations, but it is still sensitive to noise, and especially to the points that are close to be outliers, but did not reach the threshold. One of the major sources of problems was, of course, the offsets, since the values before and after them differ greatly, and, after correction, the observation will be making a slight curve in lieu of the discrete jump we had at the beginning, and that we want to detect, as shown in the picture above. Once this is complete, we get a tensor of corrections to apply, stored by day and station.

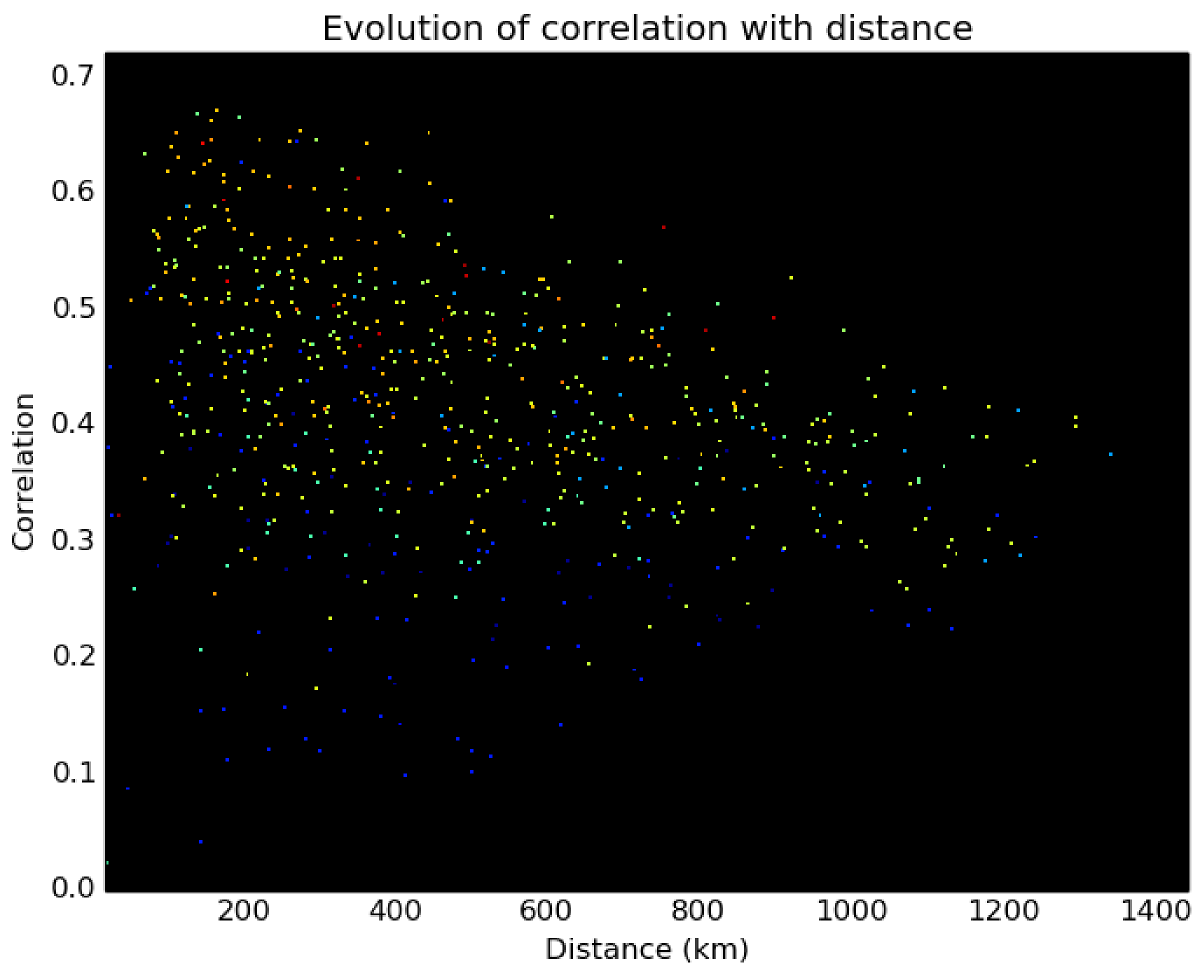
One idea I had was to put into place a spatial interpolation of these corrections, based on the following hypothesis:

- close stations should measure the same events;

- there might be a daily bias that affects the whole network.

To check if the first hypothesis I formulated was verified, I mapped the correlations between stations.

Fig 4. Correlation of the corrections between stations. On the x axis, the distances between each pair of stations, on the y axis the correlation between them. A red dot means the two stations have had a lot of days of observation in common, a blue one that they have had few.



We observe that we have a significant correlation of the corrections between close stations that have a lot of days in common, and that can be used to isolate and remove regional systematic errors. I thus used a spatial splining[1] on the corrections, in order to take into

[1]Thanks to Hippolyte Mouthier and Thomas Sandri for having provided me with a simple and reliable splining program.

account the corrections of nearby stations, and to maybe be able to spot more easily geological phenomena or to further reduce the effects of the noise.

However, the results I obtained were not satisfying. This can be explained by the small number of stations I had: they were too far away from each other to be taken into account at a significant level. To try to solve this problem, I met with the geodetic team of the nearby Victoria University of Wellington, and they suggested I acquire data from all the available GNSS stations in New Zealand, and use them for the splining (my network grew from 39 stations to nearly 300): with a denser network, the effects should be more sensible. But the results were still not good: some stations were adding enormous deviations from average to the others (up to 30 metres), and I did not manage to find out which ones, neither did I spend time on it, since spatial splining was not producing good results anyways.

In the end, I had spent around a month on this smoothing program, but all of the processes I made were extremely slow and complex, and still gave poor results. Time was running short, and I consequently decided to reject all I had done so far on this matter, smoothing and removing outliers alike, and I went for a

4th method: median smoothing. I did not use anything to deal with the outliers, and proceed directly to the smoothing on the raw data. Instead of replacing one point by the average of one one-week window centred on it, I replaced it with the median of said window. This way, most outliers are deleted, and the smoothing still gets rid of noise. This process is quite brutal, since it applies sometimes very big corrections, but it ensures a quite good smoothing, and does not smooth the edges of the offsets. It can slightly displace them, however. All things considered, this method is very effective (maybe too much...), very easy to understand and implement, and has a very short computation time. This is why I chose to stick to it, in the end. If I had more time, I believe it would be worth further investigating options for cleaning the time series.

Detecting events and adding them to the model

We now have data ready to be processed. We need to find the seismic events within it, and to add them to the model of the station position.

The existing software

Before I arrived in LINZ, there already existed software to model the time series of each station. This software, called spm_editor, allowed for the creation of a complete model for each station: once it has loaded a station file (see annex 1 for example), it creates a model for it, fitting data via least-squares, and detecting annual and semi-annual terms. It also has a few subfunctions which allow to add to this model an offset, a ramp, or a slow slip, given their date, and optionally their size. It also allows to reject points, which was typically used for outliers. Once all events have been added, it can fit the data, i.e. compute a new model taking all these new parameters into account. By default, it can freely change the magnitude of all events, and the length of ramps & slow slips, to best fit the data. This can be, however, changed, and one can fix parameters or on the contrary allow them to be fitted too. For example, if the date of an event is known only roughly, allowing it to be fitted too can improve the model. Obviously, since fitting dates adds non-linear terms into the least-square estimation, it takes a lot more computation time and can fail to find a solution at all.

This software comes with an interface, for a visual detection of events, described in annex 2.

How the jumps are detected

In a first attempt to detect the jumps, I decided to code a short program in R [5, 6]. I chose this language because of the huge number of libraries it has, and especially the jump detection ones, of course. Indeed, R is well known for its thousands of user-created libraries, amongst which a reasonable amount are meant to deal more easily with time series.

There exist ways to launch R via Python, which was useful to launch this code via my main function after the smoothing is done. Passing variables from one language to another was, however, not that simple, and I chose the easiest way to do it, albeit surely not the most efficient: I simply wrote the dates of the jumps I found with R in a file, that I read later with Python, and I added the corresponding offsets to the model.

The most complete jump-detection R library I found was changepoint [7], and it is this one that I used. It includes various offset-detection methods that I did not initially try to understand in detail. I chose an observation file that contained all of the phenomena I wanted to detect (wide slow slips, short slow slips, and jumps), and tried all of the available methods on it. I was then able to sort these methods by efficiency and computation time. Most of them detected either only one jump, or thousands. Only a few detected a reasonable amount of offsets, but even then there was still a tendency to overdetection. I chose the method that looked like the most effective (meanvar, with an asymptotic penalty, a penalty value as low as possible, the Binary Segmentation method, a maximum number of detected points of 50 to be sure to get them all, and assuming a normal distribution of the data).

All of these jump-detection methods are meant to only find clearly-defined jumps, and my time series were either too noisy or too smoothed, thus making these methods struggle.

To try to remove the less significant jumps, I tried to add them all to the model, fit, and remove all the jumps that were smaller than a certain threshold. The results were not too bad, since they fitted quite well the model, but they:

- lacked rigor, since this method is basically equivalent to adding a jump of each observation, fit the model, and remove the smallest ones;

- still had overdetection problems;

- also had underdetection problems, with visible offsets not being detected.

Dealing with slow slips

From the results I obtained, it appears that the slow slips are detected as a succession of small jumps. I thus tried to identify these successions of small jumps, and to replace them with slow slips.
I created lists of offsets going the same way (increase/decrease in the value) or with very weak values, and separated by less than 4 months from one offset to the next one (a value I chose based on observations of the time series). Once a series of offset is completed, I try to replace them all by a slow slip, and then check if the slow slip fits the data better. Should it be the case, the sequence is replaced with the slow slip.

Nevertheless, we still have trouble detecting the slow slips, as explained further in the middle report, in annex 4 (in French): maybe not all the offsets are part of the slow slip, and the closest ones to the edge of the list should be removed. But trying to find which ones would take a lot of computation time.

A way to improve my results was suggested by the geodetic team of the Victoria University of Wellington: using multiple jump-detection algorithms and see what jumps they give: the more methods that identify a particular date the greater the chance that something really happened at that date. But this is complicated to implement, since different algorithms are likely to find close but different dates for the same event. It requires clustering dates, complex analysis, and still does not handle slow slips. I chose not to try it and went on a completely different approach.

SECOND PART: CREATING A PROGRAM TO DETECT SISMIC EVENTS

After the relative lack of success of my first program, mainly due to all the “already coded” jump-detection methods overdetecting a lot, I decided to code my own. According to Xavier Collilieux, it appears that the JPL_STP1 algorithm, described by Gazeaux J. [8] and developed by NASA's Jet Propulsion Laboratory (JPL), provided an effective and easy-to-implement method, and it is consequently this one I chose to work with.

The JPL_SPT1 algorithm

Gazeaux J. provides a brief explanation how the algorithm works. It works with “residuals” time series, i.e. time series cleared of annual and semi-annual signals, as well as of their trend. It works separately on each component and tests each day for the existence of a jump in a mini-series of specified width centred upon the candidate point. The algorithm fits a line on each half of the mini-series, with the same velocity, as well as one going through the whole mini-series to test against. For each point, the fit is repeated for windows of width ranging from 20 points to around 200. This is based under the assumption that the presence of an offset does not depend of the window size. As described in the paper, the window size should have the exact number of data points to be complete, meaning that if there was missing data, data further away would be taken to compensate. In my approach, I chose to consider the window width as the maximum number of days away from the central point data could be fetched. Should there be missing data, we would have less points, and work with that, because it did not make sense to me to try to reach further away data, since some of the holes in my data can last for dozens of days, and the extra points would in these cases not add anything but outliers. The limit in the size of the window is there because with too big windows, there is an increased risk that they contain several offsets.

Once the fitting is done, an F-Test on the squared residuals of the fitting is used to see how much better the fitting under a jump assumption is, compared to the fitting without jump. According to the paper, “candidate offsets with a minimum strength metric [a function of the estimated magnitude of the jump, the WRMS, and the probability that the variances with or without jump fits are not significantly different] are retained in the candidate list.” However, this metric function seems to be highly empirical, and I did not manage to get information about it, since no NASA document dealing with the JPL_STP1 algorithm has been made available to the public.

When the metric for all points has been calculated, only the one with the highest metric is selected, a jump is added to the model at this date, and the residuals are recalculated. The process is reiterated until no jump significant enough is found. Finally the model is cleaned by discarding all the jumps smaller than a certain threshold (0.75mm in the paper).

My implementation

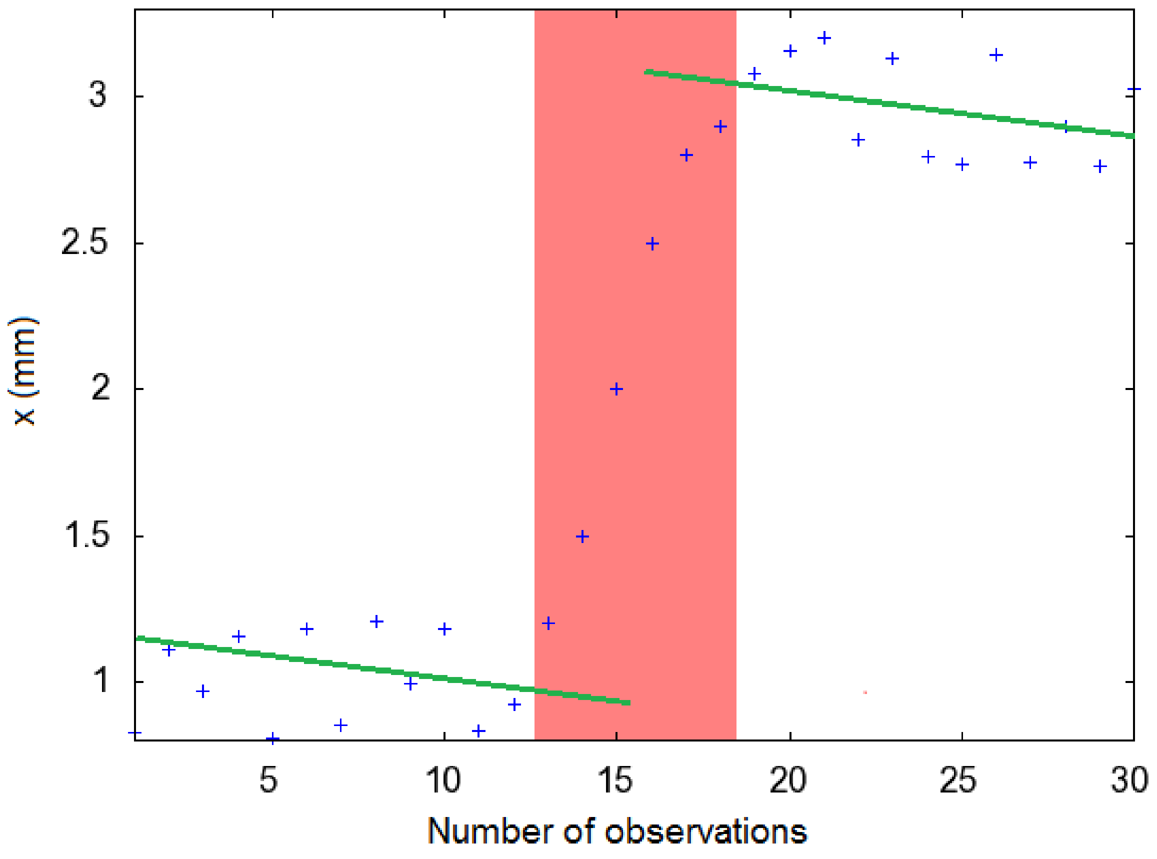
Brief presentation

My final implementation differs from the JPL_STP1 algorithm. This is mainly due to the heavy presence of slow slips in my data, which we can nevertheless theoretically be detected with a shift in the size-changing sliding windows, with a method I thought of, and that will be detailed in this paragraph. Where, in the original JPL algorithm, the different windows were used to confirm the detection of a jump, they are here used to detect slow slips.

I am working with smoothed data, cleaned of its outliers. The slight slope generated by the smoothing on the sides of the jumps should not be problematic, since it will only transform jumps into very sharp slow slips, both of which this detection method is capable of handling.

The key to dealing with smoothed offsets and slow slips is dealing differently with the data in the centre of the window than with the rest. My first idea was to simply remove the centre of each mini-series, in order to mask the eventual slope, and fit this holey data. The initial size of the gap in data was 1/5th of the window size, in order to still have enough points to have a meaningful fit, but also to remove enough observations to hide the slopes.

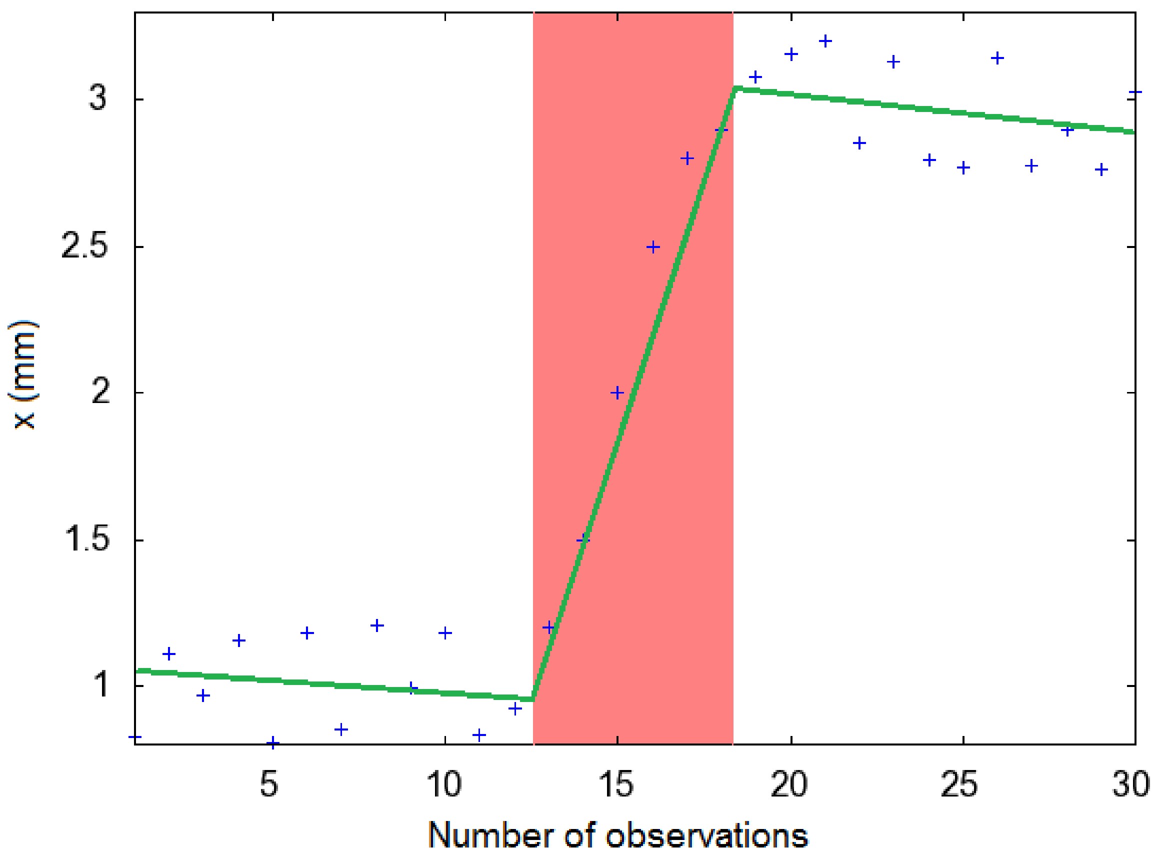
Fig 5. The data into the red zone will not be taken into account for the fitting. This way, the slow slip is artificially transformed into a neat jump. The parallel green lines are the result of the fitting.



This, combined with the different window sizes, should be able to detect slow slips, and treat them as jumps, in a first approximation. However, on smaller windows, the small amount of data leaves this system very sensitive to noise. Moreover, due to the offset being found in a zone without data, its positioning is not very precise, and it might be found a few days away from the real event.

The next idea was to keep the gap in the middle, but to try fitting not two lines anymore, but three: the edges would stay like they were, and I would be looking for two parallel lines with an offset, and between them a ramp.

Fig 6. The data in the red zone is now fitted with a ramp.



This way, all the data is kept, and our model is closer to the theoretical slow slips we are looking for. Along with the different window sizes, we have different ramp sizes (1/5th of the window, 1/20th, 1/60th...). This allows for a detection of wide slow slips (with a long ramp), as well as of offsets (with a ramp of length 1 or 2 days). We will go through the whole process we are about to describe once for every possible ramp size. Which means, for a given ramp size, we will find all the possible events with all the available windows sizes, and once we are done, we go on to the next ramp size to try finding more. Since the bigger slow slips are, the harder they are to detect, and the less significant they are, we will proceed with decreasing ramp sizes. This will allow detection of the wide slow slips first, before they could possibly be masked out by a smaller one or an offset that would be detected at the same date.

Statistical basis

For each day, I use windows centred upon said day to generate a mini-series, then fit both its halves and the middle (i.e. with an offset/slow slip), as well as the whole mini-series (i.e. without offset), and see how much better the fitting with an offset is. Our null hypothesis is that there is a ramp in our data and that our unrestrained model (with the ramp) is the valid model. It will be tested against the hypothesis that the data can be fitted as well with a single straight line (restrained model).

For that, I use an F-Test, with the following formula:

$$F=\frac{(ssr_R-ssr_U)\times dof_U}{3\times ssr_U} \qquad (2)$$

where:

- ssr_R and ssr_U are the sums of the least square residuals for the restricted and the unrestricted model, respectively;
- dof_U is the number of degrees of freedom for the fitting of the unrestrained model;
- the 3 comes from the difference of degrees of freedom between the unrestrained and the restrained model, as we are fitting all dimensions (east, north, and up) at once.

If the restrained model doesn't fit as well as the unrestrained model then the difference in the residuals, and hence the F value, will be large. So a large F value implies that we need the ramp to fit the data at this location.

Under the assumption that our data follows a normal distribution, F should follow a $F_{3,dof1}$ distribution, and we use Python built-in functions to give us the probability of such a variable reaching the value F.

The results of the F-Test, the significance of the jump, and its magnitude (given by the parameters of the least square fitting) are stored in tensors, and will be used later.

Selecting the events

Once all days have been computed, we will look at the biggest significance that has been found. In case of very close or equal significance (typically 99.9999999% for the first jump), we will look at the result of their F-Test. Because the first jump is often very significant, its significance using the F probability distribution becomes very insensitive between days around the event. It is more reliable to use the F statistic directly. The day where this reaches its maximum value is likely to be the time of the offset, or slow slip, we are looking for. We then check that our proposed offset has a big enough magnitude (greater than 2mm horizontally or vertically), and that is not too close to an event we have already found. Events are not allowed to be closer than 20 days to another event found previously. In a first approximation, the magnitude we use is computed with the parameters of the least squares estimation: it is the difference between the two parallel lines we fitted.

If our proposed offset did not fulfil all of the aforementioned conditions, it is marked as not to be computed again, its significance is set to 0, and we then look at the next bigger significance, until we find an event. If we don't find an event with significance over the minimum significance (which we fixed at 98%) then we go on to the next smallest ramp size.

If our offset did fulfil the conditions, it is then added to a copy of the model (model2), and its magnitude recalculated using a least squares estimate. Its magnitude is then tested again, and we have three different possibilities: if at least one of the components of the jump is bigger than the threshold, we keep it. If all components are smaller than the threshold, but at least one is bigger than half of said threshold, we dismiss the offset and continue our search with the next most significant one. It none of the components reaches half of the threshold, the offset is rejected and we end the search for this iteration, as we assume that the other jumps, being less significant, will not be big enough either.

If the offset has been kept, we then create yet another copy of the model (model3), add a slow slip centred on the date we have found, and fit it while allowing its length to change. If the slow slip length is bigger than a certain threshold (generally 4 days), we keep the model3 as the new model. Otherwise, we consider that we are dealing with an offset rather than a slow slip, and keep model2 as our new model.

Once this is done, we compute the residuals between the new model and the observations, mark the days around the event we just found as not be computed again in order for the same event not to be found again, and we reiterate all the computations for this ramp size, until no jump significant enough, or big enough, is found. Once no more jumps are found, we reiterate the whole process with another ramp size, until we have done them all.

With this algorithm, for 39 stations working, for some of them, since early 2001, and recording one observation per day, it took a little less than two hours to run on typical desktop computer.

Problems encountered

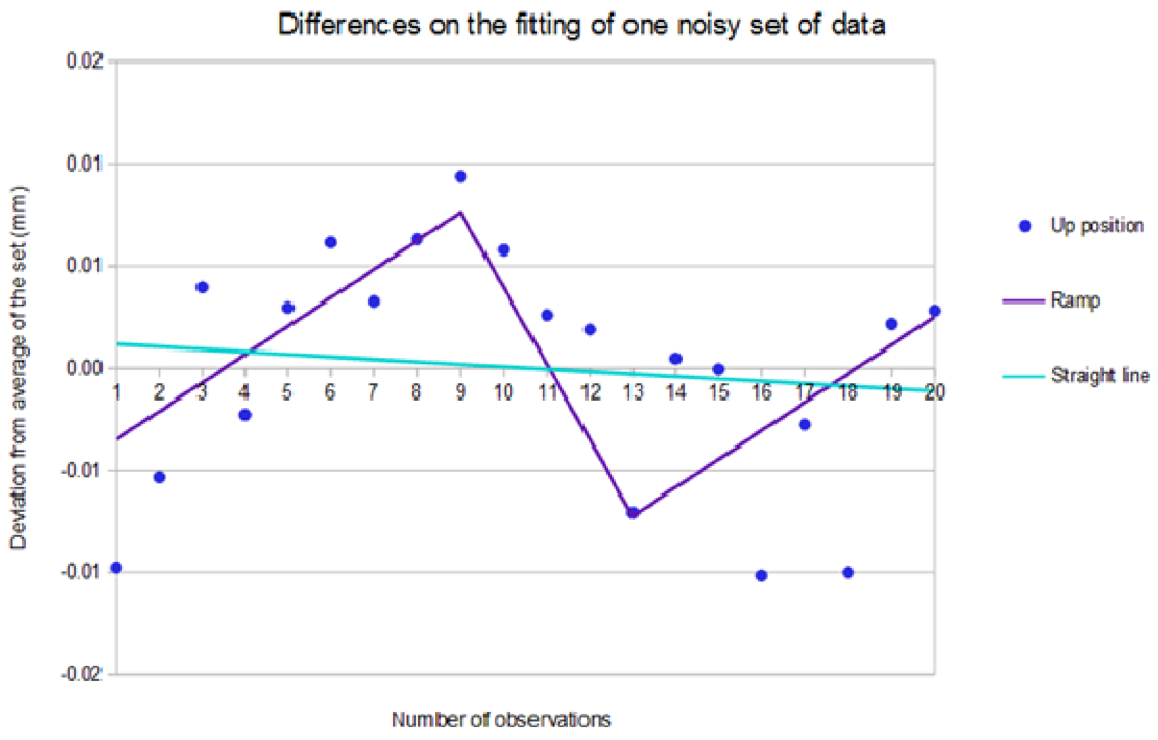
- To quantify the improvement in the fitting between the “two halves and a ramp” and the “straight line” ones, the only statistical tool we have at our disposal is the F-Test. However, this test relies on statistical assumptions about the data, all of which may not be verified in our case. First of all, there are little chances that the residuals we are working on follow the required normal distribution, due to the presence of outliers, and of the nature of GNSS time series [9, 10]. They should also be strictly independent, and we cannot guarantee that either, since there might be some systematic error affecting the observations.

This has an extremely annoying effect, which is to put the confidence in having an offset this day to above 99.9%, every day. Due to the asymptotic nature of the F probability distribution function, there is a huge difference in the F-Test value between a 99.9995% confidence, and a 99.99999% one, and we thus use the values of the F-Test to discriminate between two days is the confidences are equal, or very close. This is not exactly correct, since the relation between the result of the F-Test and its significance also relies on the number of observations taken into account, and thus on the size of the windows, which should make comparisons between F-Test results from two different windows meaningless. However, if the number of observations gets above around 20, its value will typically not change anything, a bit as if anything bigger than 20 was considered as infinity. Moreover, we do not really have any other choice. If we still have a tie between two dates for which F-Test results are exactly identical (and that happens; rarely, but that happens), we take the first day in chronological order.

- Due to the forbidding of overlapping of events, one event found early with a huge ramp function might turn out to be more significant later with a smaller ramp, but would not be taken, since there is already an event close to its date. Typically, it would happen with very close offsets being modelled as slow slips early in the run, and not being able to be taken back as offset later when the windows get smaller. To overcome that problem, we have changed the rules for overlapping events: events still cannot be found within 20 days of an offset, but they can be arbitrarily close to slow slips. This enables overlapping slow slips, slow slips within slow slips, and jumps within slow slips, all of which are possible geological phenomena.

- Even if data has been smoothed, there still is noise remaining, and, with small windows, this noise can be mistaken for jumps.

Fig 7. Fitting one set of points taken at random from the data with either a ramp (in purple) or a straight line (light blue). We can clearly see the influence of the very few points under the average on the fitting of the ramp.



Results

I have no real tool to see how good the model I fitted is. With the smoothing, the original time series are heavily modified to remove outliers and noise, and I have found no way to qualify the changes I made or to measure how good they are: an outlier is a vague notion, subject to user-defined threshold, and did not try to measure quantity of noise removed. Then, trying to compute standard deviations or means of differences between observations and model would not make sense: the original time series still contain outliers which will contaminate the results. I guess I could assess the model against the modified time series, since the whole reason for modifying it is to get it to better represent the signal and discard the noise. But the main problem for assessing the models is that I do not have any measure of what slow slips and jumps should be detected.

One way to better assess my methodology could be, for example to write a program to generate time series with random noise, slow slips, and offsets, and then see how well the algorithm is able to detect them: this way, I would know where the events happen, and I could make proper statistics. I did consider this, but, again, lacked time to do it.
Some examples of the results I obtained are shown in annex 3.

So, the results I have obtained, i.e. the models for each station, seem good, but the only way I have to check it is to plot the model alongside of the original and modified data, and perform visual inspection. And from what we have, we can see a very good detection of jumps, and, most of the time, a great management of slow slips too, which is very satisfying given the complexity of the problem, and the scarce amount of literature on slow slip detection.

CONCLUSION

On a technical point of view, I think that this internship was a real success, and I was not expecting to get results nearly as good as those I have. Early in the internship, I was hoping to manage to deal with the offsets before the end, and, if I had time, maybe look at how to detect outliers. Creating a routine to detect them both in a reasonable amount of time has gone far beyond my expectations.

Of course, this program can still be improved: its main flaw is that it has to recompute the whole model each time observations are added to one station file. One idea I had to overcome this problem is to give the possibility to the user to fit the model only for a certain range of date, and to take the events of the old model for the rest. This would make the program considerably faster, and allow “real-time” detection of events. Given the structure of my code, this feature could be easily implemented, but, once again, I lacked time.

On a personal aspect, now, this internship was obviously the longest project I had been given so far, and I was given a lot of independence, be it on the rhythm of work, the way of doing things, or even the objectives to reach. It has taught me a lot about working by myself (even if my supervisor was here when I needed him), self-teaching (on Python, statistics…), and even self-motivation, when weeks of efforts failed to produce the tiniest improvement.

On the whole, it was a very positive and rewarding experience, both on personal and professional aspects, which also permitted me to visit New Zealand and to encounter new people.

BIBLIOGRAPHY

I used most of the documents listed here just to make general ideas about a subject, and I did not always try to understand everything in them. They mostly deal with subsidiary subjects, since there is almost no documentation at all on the central and most difficult part of my research, namely the detection of slow slip events.

[1] Grubbs, F. E., *Procedures for detecting outlying observations in samples*, *Technometrics* 11 (1): 1–21, (1969). DOI: 10.1080/00401706.1969.10490657

[2] Ben-Gal, I., *Outlier detection*, In: Maimon O. and Rockach L. (Eds.) Data Mining and

Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers, Kluwer Academic Publishers, 2005, ISBN 0-387-24435-2

[3] Analytical Methods Committe, *Robust statistics: a method of coping with outliers* (2001)

[4] Leys, C., et al., *Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median*, Journal of Experimental Social Psychology (2013), http://dx.doi.org/10.1016/j.jesp.2013.03.013

[5] Paradis, E., 2002, *R pour les Débutants*, Montpellier, University of Montpellier. URL: http://cran.r-project.org/doc/contrib/rdebuts_fr.pdf

[6] R Development Core Team, *R: A language and environment for statistical computing. R Foundation for Statistical Computing*, Vienna, Austria, 2005. ISBN 3-900051-07-0, URL: http://www.R-project.org.

[7] Killick, R., Eckley, I.A., *Changepoint: an R package for changepoint analysis* (2011). URL: www.lancs.ac.uk/~killick/Pub/KillickEckley2011.pdf

[8] Gazeaux, J., et al., *Detecting offsets in GPS time series: first results from the detection of offsets in GPS experiment*, J. Geophys. Res. Solid Earth, 118, 2397–2407 (2013) doi: 10.1002/jgrb.50152

[9] Rodionov, S. N., *Use of prewhitening in climate regime shift detection,* Geophysical Research Letters, Vol. 33, L12707 (2006), DOI: 10.1029/2006GL025904

[10] Williams, S. D. P., *The effect of coloured noise on the uncertainties of rates estimated from geodetic time series*, Journal of Geodesy (2003) 76: 483–494, DOI: 10.1007/s00190-002-0283-4

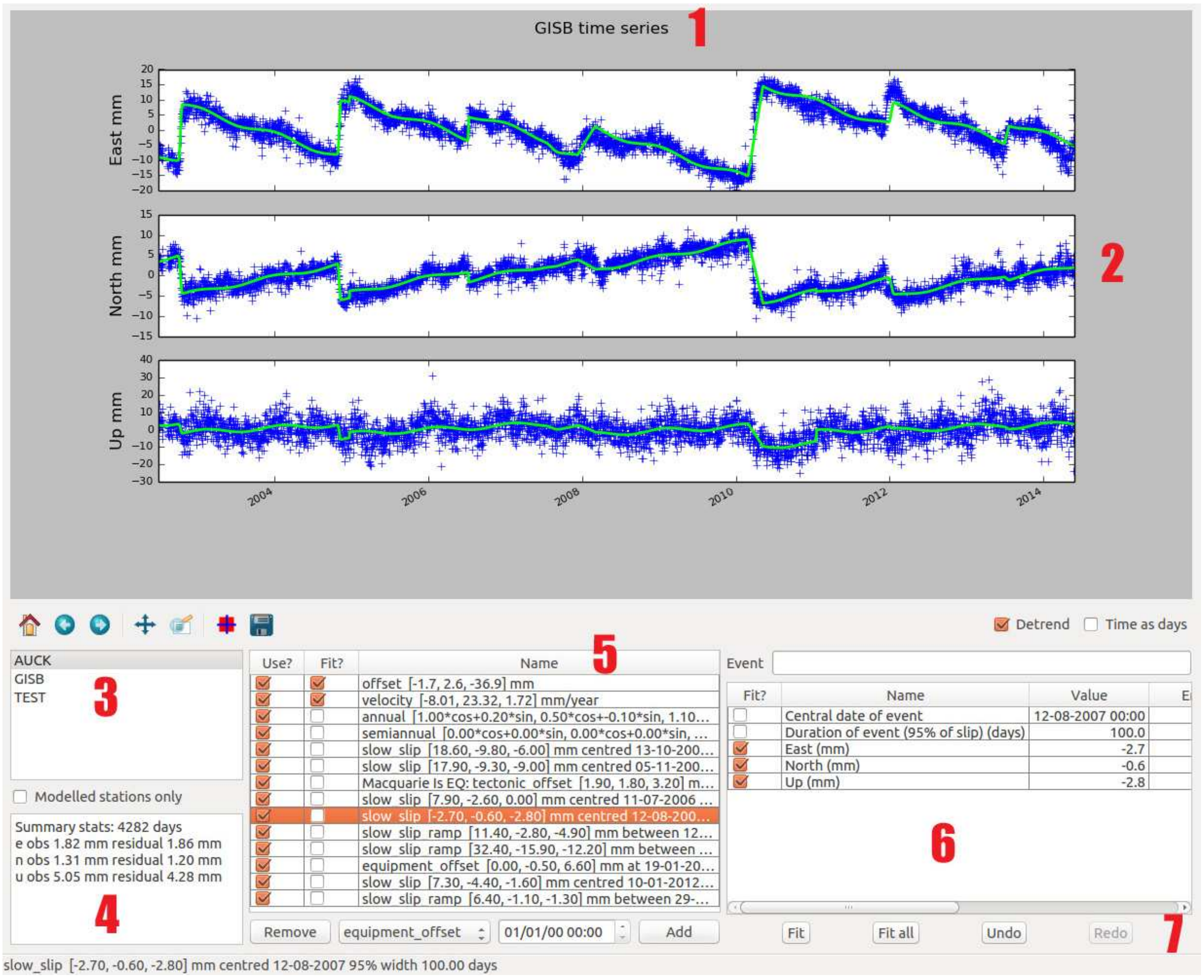
ANNEX 1: EXAMPLE OF OBSERVATION FILE

Here is an extract of the observation file for the WGTN station of Wellington.

```
name   epoch  x      y      z      flag
WGTN   2000-01-02T12:00:00 -4777269.3987      434270.0649  -4189484.5488      A
WGTN   2000-01-03T12:00:00 -4777269.3959      434270.0665  -4189484.5465      A
WGTN   2000-01-04T12:00:00 -4777269.3941      434270.0688  -4189484.5450      A
WGTN   2000-01-05T12:00:00 -4777269.4011      434270.0640  -4189484.5575      A
WGTN   2000-01-06T12:00:00 -4777269.3953      434270.0655  -4189484.5517      A
WGTN   2000-01-07T12:00:00 -4777269.4024      434270.0610  -4189484.5591      A
WGTN   2000-01-08T12:00:00 -4777269.4006      434270.0640  -4189484.5557      A
WGTN   2000-01-09T12:00:00 -4777269.4014      434270.0645  -4189484.5493      A
WGTN   2000-01-10T12:00:00 -4777269.4062      434270.0626  -4189484.5590      A
WGTN   2000-01-11T12:00:00 -4777269.4052      434270.0635  -4189484.5542      A
WGTN   2000-01-12T12:00:00 -4777269.3956      434270.0651  -4189484.5499      A
WGTN   2000-01-13T12:00:00 -4777269.4075      434270.0623  -4189484.5595      A
WGTN   2000-01-14T12:00:00 -4777269.3980      434270.0650  -4189484.5498      A
WGTN   2000-01-15T12:00:00 -4777269.3959      434270.0662  -4189484.5469      A
WGTN   2000-01-16T12:00:00 -4777269.3982      434270.0673  -4189484.5520      A
WGTN   2000-01-17T12:00:00 -4777269.4006      434270.0669  -4189484.5488      A
WGTN   2000-01-18T12:00:00 -4777269.4006      434270.0651  -4189484.5514      A
WGTN   2000-01-19T12:00:00 -4777269.4085      434270.0663  -4189484.5551      A
WGTN   2000-01-20T12:00:00 -4777269.4043      434270.0660  -4189484.5526      A
WGTN   2000-01-21T12:00:00 -4777269.4138      434270.0683  -4189484.5607      A
WGTN   2000-01-22T12:00:00 -4777269.3841      434270.0781  -4189484.5325      A
WGTN   2000-01-23T12:00:00 -4777269.3986      434270.0809  -4189484.5395      A
WGTN   2000-01-24T12:00:00 -4777269.3962      434270.0749  -4189484.5435      A
WGTN   2000-01-25T12:00:00 -4777269.4023      434270.0675  -4189484.5484      A
WGTN   2000-01-26T12:00:00 -4777269.4010      434270.0700  -4189484.5513      A
WGTN   2000-01-27T12:00:00 -4777269.3972      434270.0695  -4189484.5471      A
WGTN   2000-01-28T12:00:00 -4777269.3946      434270.0685  -4189484.5456      A
WGTN   2000-01-29T12:00:00 -4777269.3962      434270.0728  -4189484.5443      A
WGTN   2000-01-30T12:00:00 -4777269.3875      434270.0699  -4189484.5345      A
WGTN   2000-01-31T12:00:00 -4777269.3985      434270.0689  -4189484.5451      A
WGTN   2000-02-01T12:00:00 -4777269.4014      434270.0706  -4189484.5497      A
WGTN   2000-02-02T12:00:00 -4777269.4030      434270.0678  -4189484.5563      A
```

ANNEX 2: DESCRIPTION OF THE INTERFACE OF THE VISUAL EVENTS DETECTION SOFTWARE

Here is a commented screenshot of the software that was used in LINZ for visual detection of tectonic events. It shows most of its possibilities.



AUCK
GISB
TEST

☐ Modelled stations only

Summary stats: 4282 days
e obs 1.82 mm residual 1.86 mm
n obs 1.31 mm residual 1.20 mm
u obs 5.05 mm residual 4.28 mm

Use?	Fit?	Name
✓	✓	offset [-1.7, 2.6, -36.9] mm
✓	✓	velocity [-8.01, 23.32, 1.72] mm/year
✓	☐	annual [1.00*cos+0.20*sin, 0.50*cos+-0.10*sin, 1.10…
✓	☐	semiannual [0.00*cos+0.00*sin, 0.00*cos+0.00*sin, …
✓	☐	slow slip [18.60, -9.80, -6.00] mm centred 13-10-200…
✓	☐	slow slip [17.90, -9.30, -9.00] mm centred 05-11-200…
✓	☐	Macquarie Is EQ: tectonic offset [1.90, 1.80, 3.20] m…
✓	☐	slow slip [7.90, -2.60, 0.00] mm centred 11-07-2006 …
✓	☐	slow slip [-2.70, -0.60, -2.80] mm centred 12-08-200…
✓	☐	slow slip ramp [11.40, -2.80, -4.90] mm between 12…
✓	☐	slow slip ramp [32.40, -15.90, -12.20] mm between …
✓	☐	equipment offset [0.00, -0.50, 6.60] mm at 19-01-20…
✓	☐	slow slip [7.30, -4.40, -1.60] mm centred 10-01-2012…
✓	☐	slow slip ramp [6.40, -1.10, -1.30] mm between 29-…

Event

Fit?	Name	Value	E
☐	Central date of event	12-08-2007 00:00	
☐	Duration of event (95% of slip) (days)	100.0	
✓	East (mm)	-2.7	
✓	North (mm)	-0.6	
✓	Up (mm)	-2.8	

slow_slip [-2.70, -0.60, -2.80] mm centred 12-08-2007 95% width 100.00 days

(1): name of current station;

(2): graphics showing the data (blue) and the current model (green). It can also display the points rejected as outliers by the operator, as red crosses;

(3): list of loaded stations;

(4): global statistics about the current model;

(5): list of the events detected by the operator. Each of these events can be, or not, taken into account into the model (first column), adjusted to best fit the model (second column), and is identified by a brief description (third column). Within these events are at least the velocity of the station, and annual and semi-annual terms. There can also be earthquakes and slow slip events;

(6): list of the characteristics of the selected event. Each of them can be adjusted to best fit the model (first column). Characteristics of a tectonic event are its cartesian components (east, north & up), plus, for a slow slip, its central date and length, and for an earthquake, just its date;

(7): First button: removes the selected event.

Second button: possibility to add an event by either selecting its type and date, or only its type, and clicking on the graphics (2) to select where to put it.

Third button: fits the selected event.

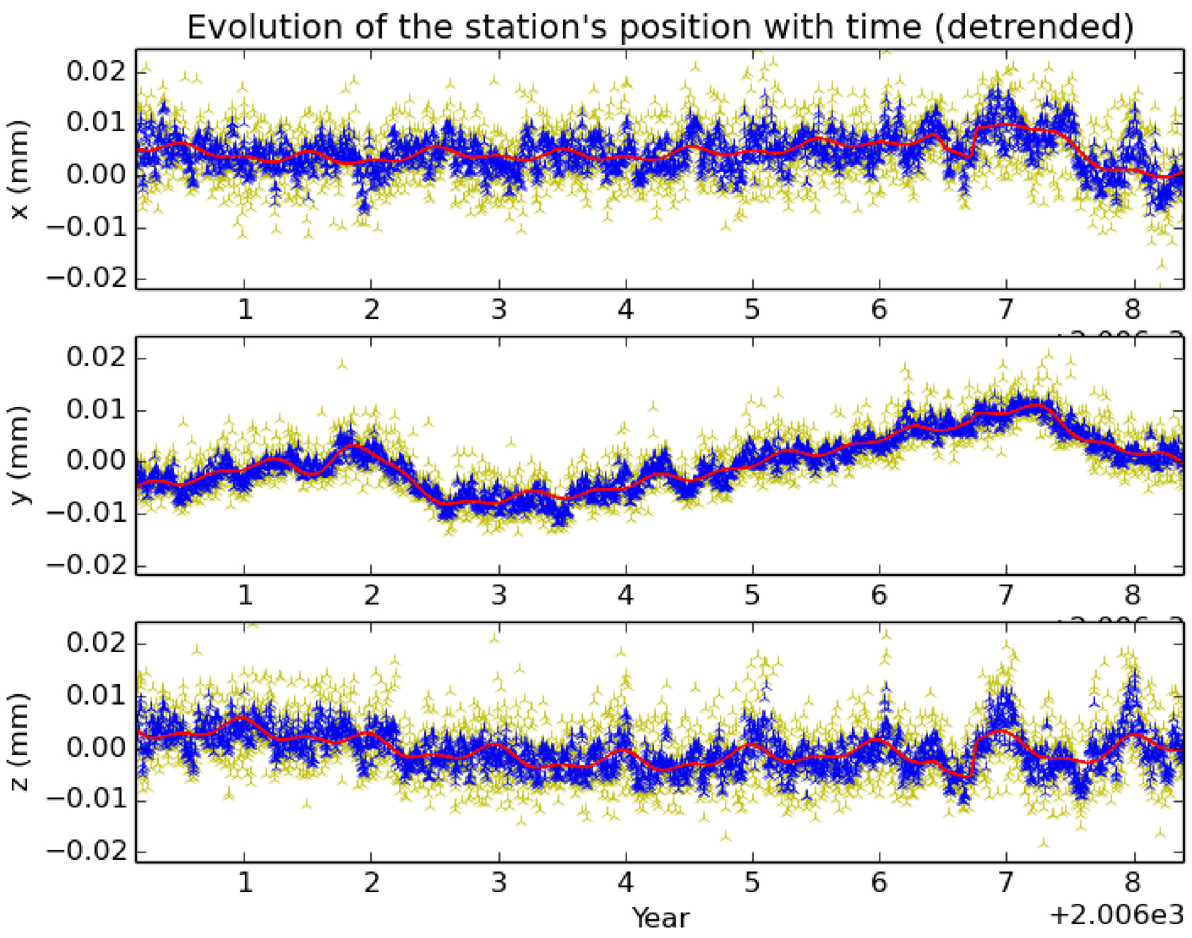
Fourth button: fits all events.

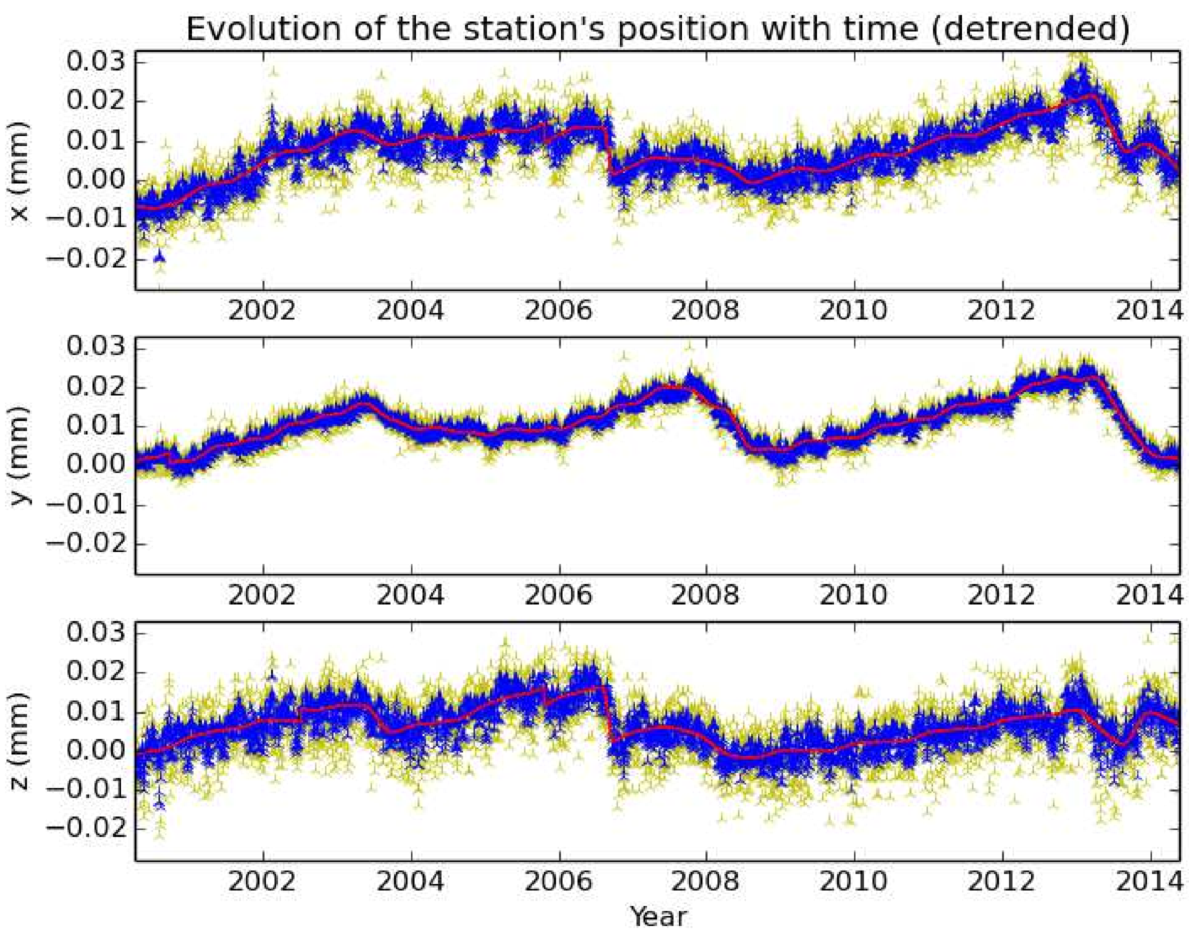
ANNEX 3: SELECTED PLOTS OF MY AUTOMATICALLY CALCULATED MODEL

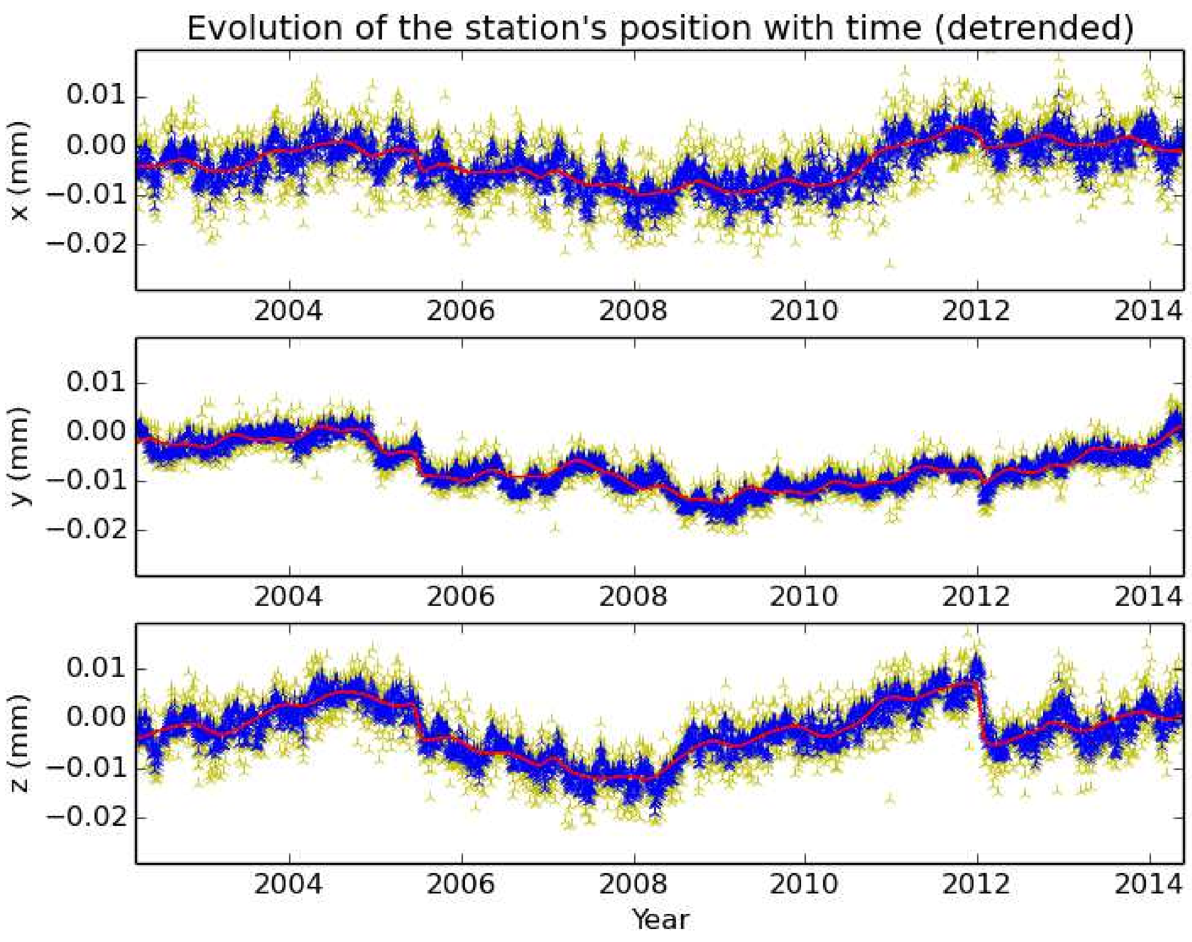
The graphics I present here are a selection of the plots of the calculated models for the station of the PositioNZ network I worked with. These plots are detrended: I roughly removed, for aesthetical purposes only, the velocity of the station, in order to better see the tectonic events. To do that, I assumed that the velocity was the director coefficient of the line passing through the first and the last observations. This assumption has been proven wrong in case of strong earthquakes, but the plots are still good enough to be meaningful.
Each graphic has been selected to show something in particular, they are all preceded by a brief explanation.

On each graphic, three sets of data are present: the original raw data in greenish yellow, the smoothed data in blue, and the final model in red.

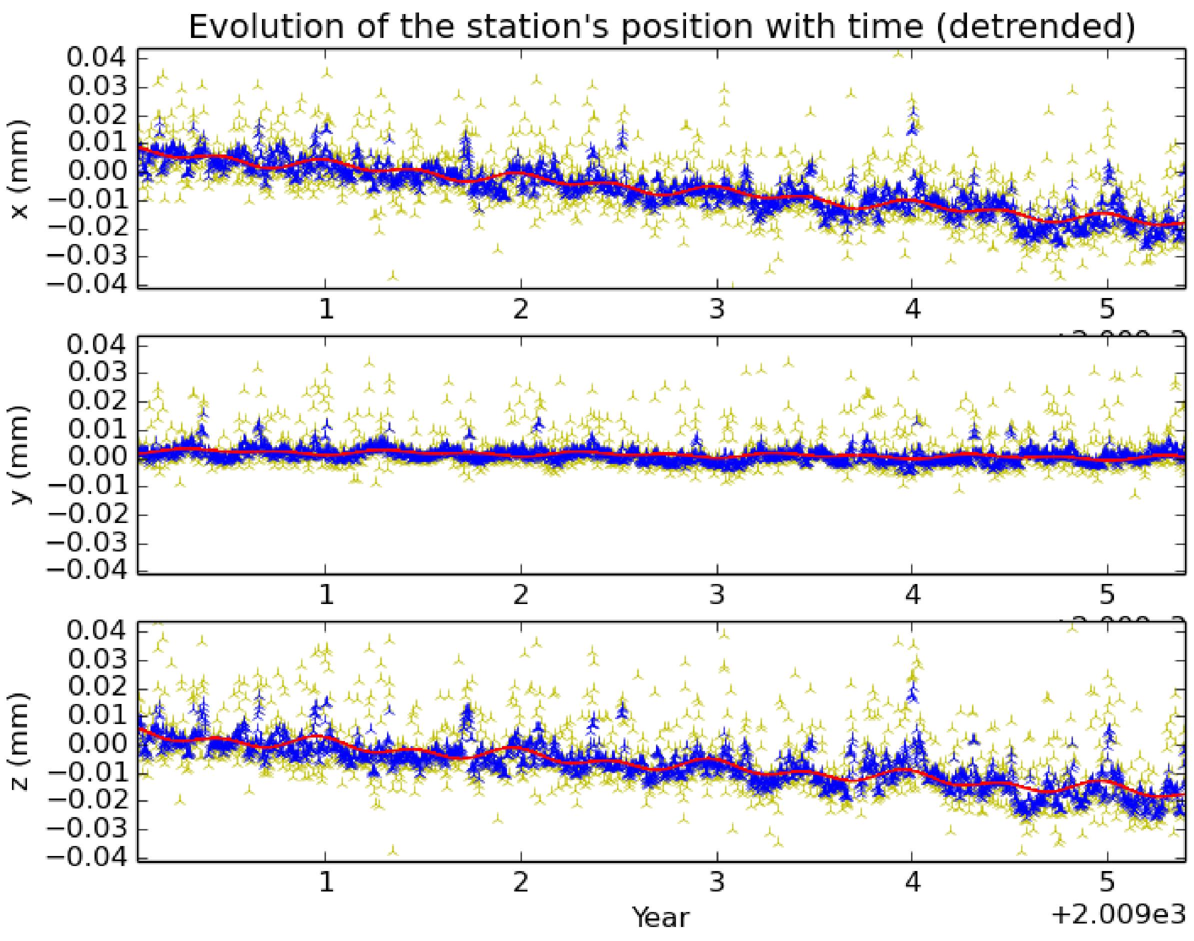
The next stations show a mix of offsets and slow slips, sometimes of small magnitude, and show of how good the fitting is, regardless of the number or complexity of events.



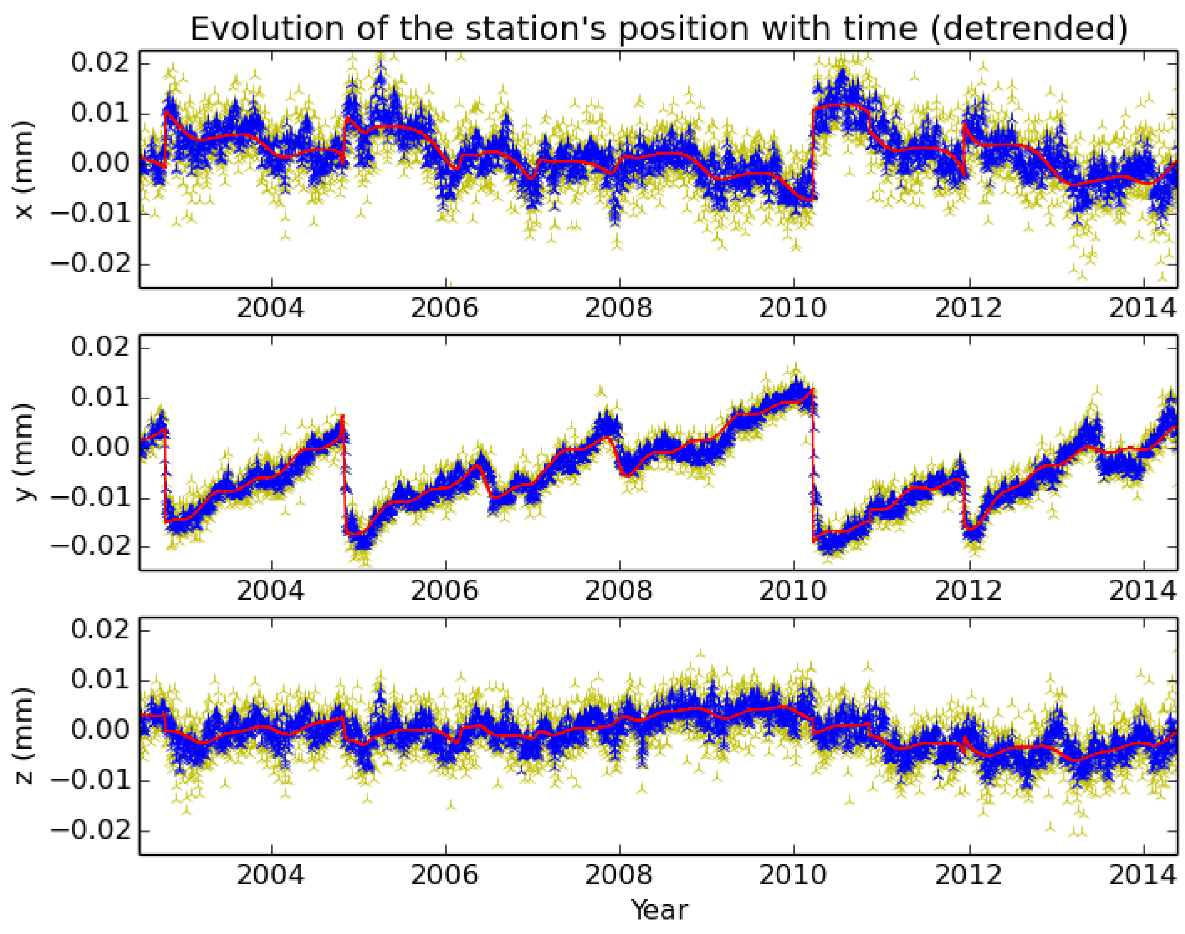

Evolution of the station's position with time (detrended)
x (mm)
y (mm)
z (mm)
0.03
0.02
0.01
0.00
−0.01
−0.02
2002
2004
2006
2008
2010
2012
2014
Year

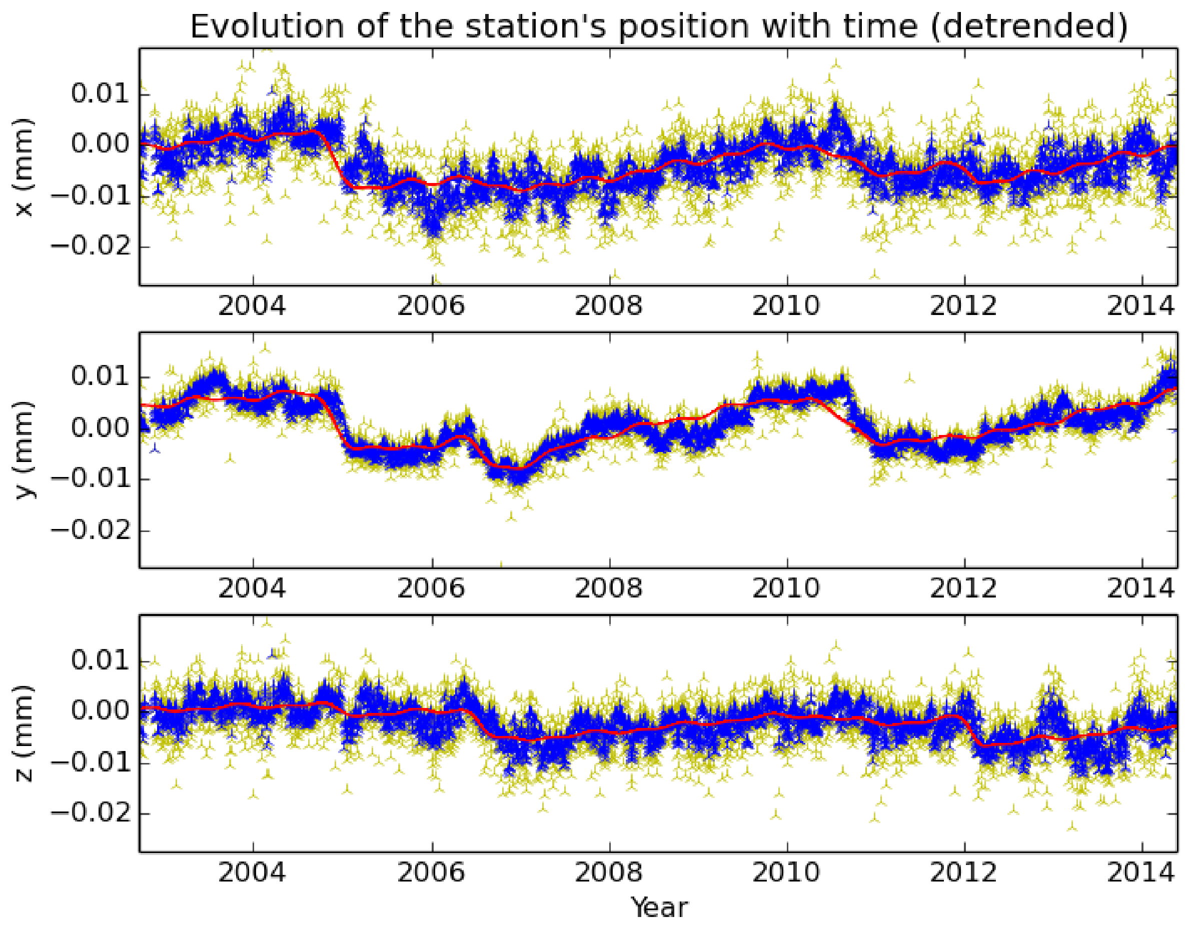

Evolution of the station's position with time (detrended)
x (mm)
y (mm)
z (mm)
0.01
0.00
−0.01
−0.02
2004
2006
2008
2010
2012
2014
Year

I chose this station to show the efficiency of the smoothing, and its capacity to work even with very noisy observations.

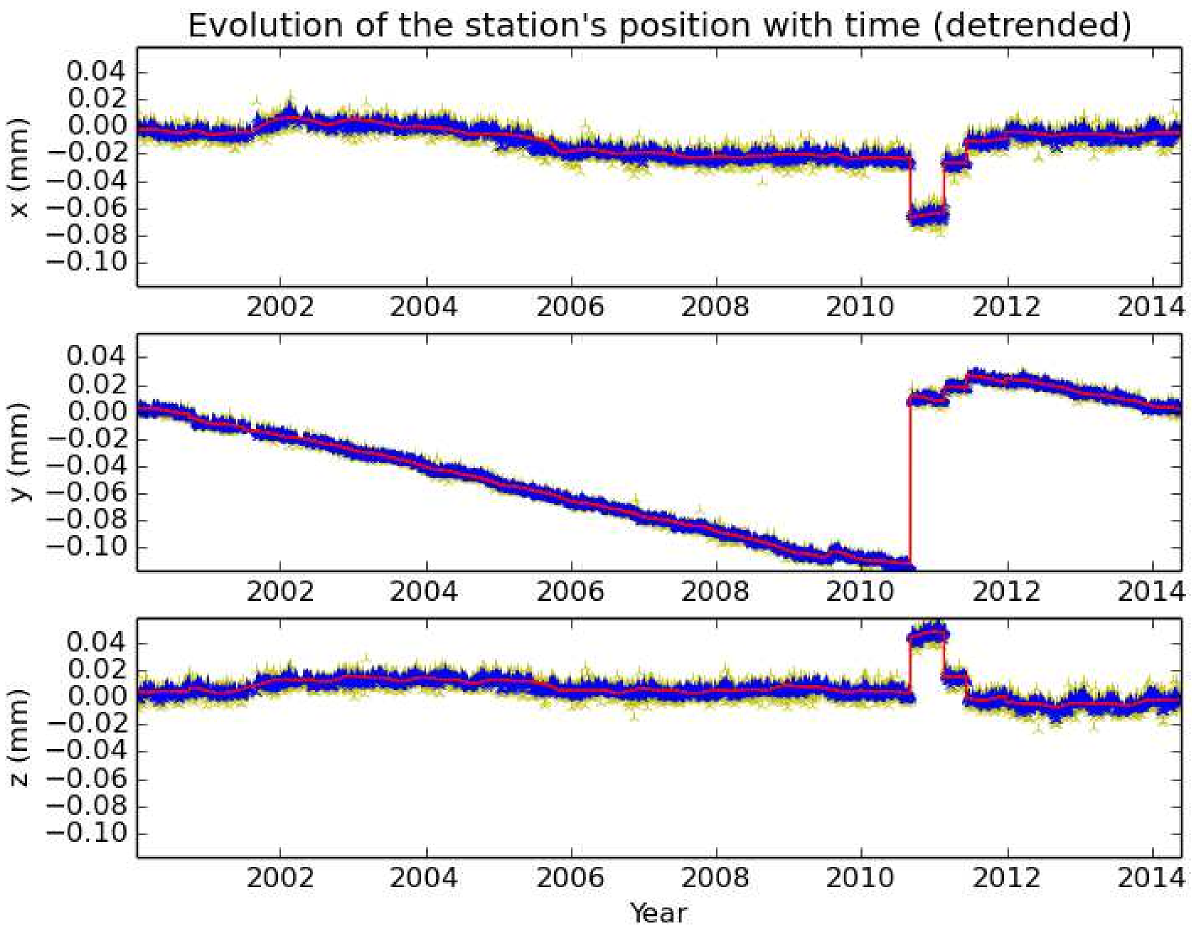


The two stations on the next page are the ones with the most numerous and biggest seismic events. They are thus extremely irregular, but still fitted quite well.


Evolution of the station's position with time (detrended)
x (mm)
y (mm)
z (mm)
0.02
0.01
0.00
−0.01
−0.02
2004
2006
2008
2010
2012
2014
Year


Evolution of the station's position with time (detrended)
x (mm)
y (mm)
z (mm)
0.01
0.00
−0.01
−0.02
2004
2006
2008
2010
2012
2014
Year

Finally, the last station I chose to show is one located in Christchurch, where the deadliest earthquake in New Zealand over the past 80 years happened in February 2011, followed by another one nearly as powerful within a few months.



ANNEX 4: MIDDLE REPORT

Point sur le stage à mi-parcours

I- Nettoyage des données & détection des données aberrantes

A- Distance à la moyenne

Pour nettoyer les données d'une station, j'effectue une boucle sur l'ensemble des journées d'observation, et je fais à chaque fois la moyenne des observations sur X jours avant, X jours après. En général, je prends une fenêtre d'une semaine de chaque côté de l'observation. Ensuite, je fais la différence entre la valeur de mon observation et la moyenne, et j'obtiens une correction à appliquer pour ramener cette observation à la moyenne.

En cas de trous dans les données, on saute la journée, et on fait la moyenne sans, on ne va pas chercher plus loin dans les observations pour avoir toujours autant d'observations à moyenner, ce qui n'aurait aucun sens.

En cas de saut, la moyenne sera faussée pour les quelques jours autour, et le saut très légèrement adouci, ce qui ne pose pas a priori de problèmes pour la suite, après essai sur des jeux de données test.

B- Suppression des données aberrantes

En parallèle du calcul de corrections, j'effectue une recherche des points extrêmes, pour que les données aberrantes n'affectent pas mes moyennes et donc mes corrections. J'ai implémenté pour le moment trois méthodes pour la détection de ces points.

Méthode 1. Par moyenne, la plus efficace pour le moment. J'effectue dans un premier temps ma moyenne sur tous les points alentour sans prendre en compte l'observation du jour même. Si l'écart moyenne-observation est trop important, l'observation est marquée comme point extrême potentiel. En effet, il est possible que l'observation soit détectée comme extrême par la présence d'une donnée aberrante proche qui affecte la moyenne. En revanche, si le point est sain, je complète simplement la moyenne avec l'observation concernée et je passe au jour suivant.

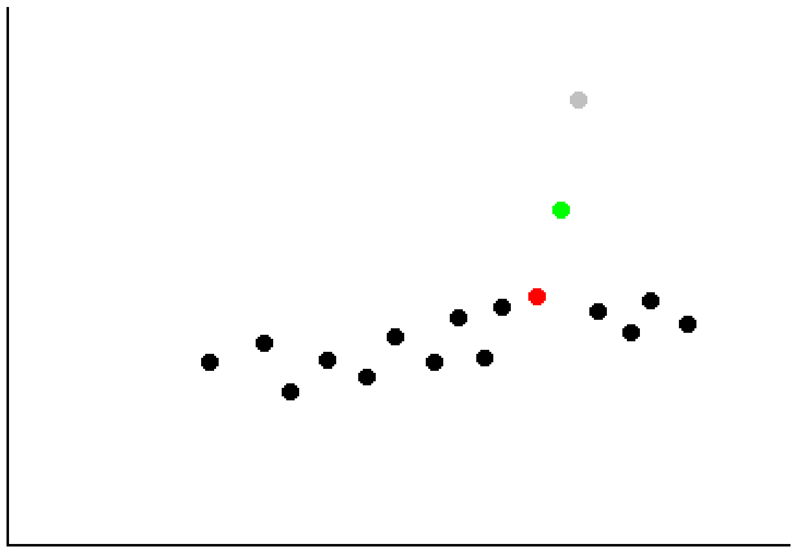


Fig 1. Un exemple de détection abusive. Le point rouge est détecté comme aberrant, à cause des deux valeurs extrêmes à côté.

Pour pallier à ce problème, une fois qu'un point A est susceptible d'être rejeté, on teste tous les points sur une fenêtre de X jours autour, si l'un d'entre eux a un écart à la moyenne supérieur à notre observation A, on cesse de s'intéresser à A pour se consacrer à notre nouveau point à potentiellement rejeter, B. Et on recommence le test, jusqu'à avoir trouvé le point le plus extrême. Il est alors purement et simplement supprimé des données. On reprend alors du point initial A, et on recommence.

Dans l'exemple de la figure 1, on détectera d'abord probablement le point rouge, puis le vert, puis le gris, qui sera supprimé. On reprend alors avec le rouge, que l'on re-détecte, puis le vert, qui sera à son tour supprimé, et on reprend encore une fois du rouge, qui n'est désormais plus suspect.

La jauge pour décider de quels points doivent être supprimés, c'est-à-dire le M tel qu'une observation est supprimée si |observation – moyenne| > M, est pour l'instant déterminé empiriquement de façon à avoir des données visuellement satisfaisantes, sans supprimer trop de points, et est fixe.

Il est fortement envisagé de la rendre variable en fonction des données.

Méthode 2. Avec la Median Absolute Deviation (MAD). Méthode décrite chez *Leys, C., et al., Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median, Journal of Experimental Social Psychology (2013), http://dx.doi.org/10.1016/j.jesp.2013.03.013*

Donne d'assez mauvais résultats pour l'instant.

Méthode 3. Par voisinage. Pour chaque point, on compte le nombre des observations de la fenêtre (X jours avant, X après) qui sont comprises dans une plage de M autour de notre observation. Les observations avec le moins de voisins seront rejetées. Une fois encore, M est choisi “à la main” par l'opérateur. L'efficacité de cette méthode varie beaucoup en fonction du niveau de bruit des données.

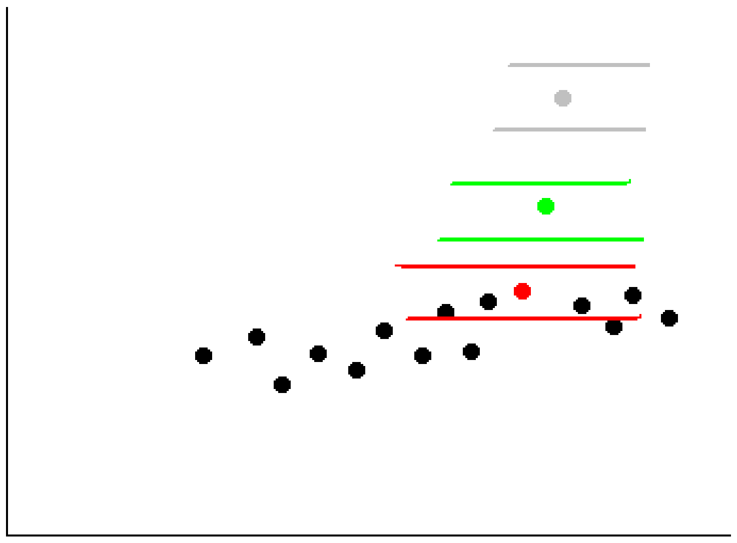


Fig 2. Méthode par voisinage. Le point rouge a 4 voisins, les points vert et gris aucun.

Remarques diverses :
- À chaque fois qu'un point est rejeté, on revient X jours en arrière dans le traitement, pour recalculer les corrections de toutes les observations qui auraient pu être affectés par ce point.
- Les traitements se font coordonnée par coordonnée : on regarde d'abord en Northing puis en Easting et enfin éventuellement en Up. Si un point est rejeté sur une coordonnée, il l'est sur toutes. On ne considère généralement pas Up, qui est beaucoup plus bruitée que les autres coordonnées
- Le traitement coordonnée par coordonnée peut introduire des problèmes au niveau de la détection. Pour un seuil de rejet fixé à 5, par exemple, un point dont les corrections en Easting Northing Up sont (0, 5.1, 0) sera rejeté mais un point à (-4.9, 4.8, 4.95) sera conservé, alors qu'il paraît “plus faux” que le premier. Peut-être faudrait-il baser la réjection sur la moyenne des corrections, mais il faudrait alors voir que faire de Up, le dépondérer, l'ignorer ?

C- Application des corrections

Une fois les données aberrantes rejetées et les corrections calculées, on va procéder jour par jour, et appliquer une spline sur toutes les corrections de la journée : chaque correction sera modifiée par celle des autres en fonction de la distance. Ceci parce que j'ai montré une corrélation relativement importante entre les corrections des stations proches.

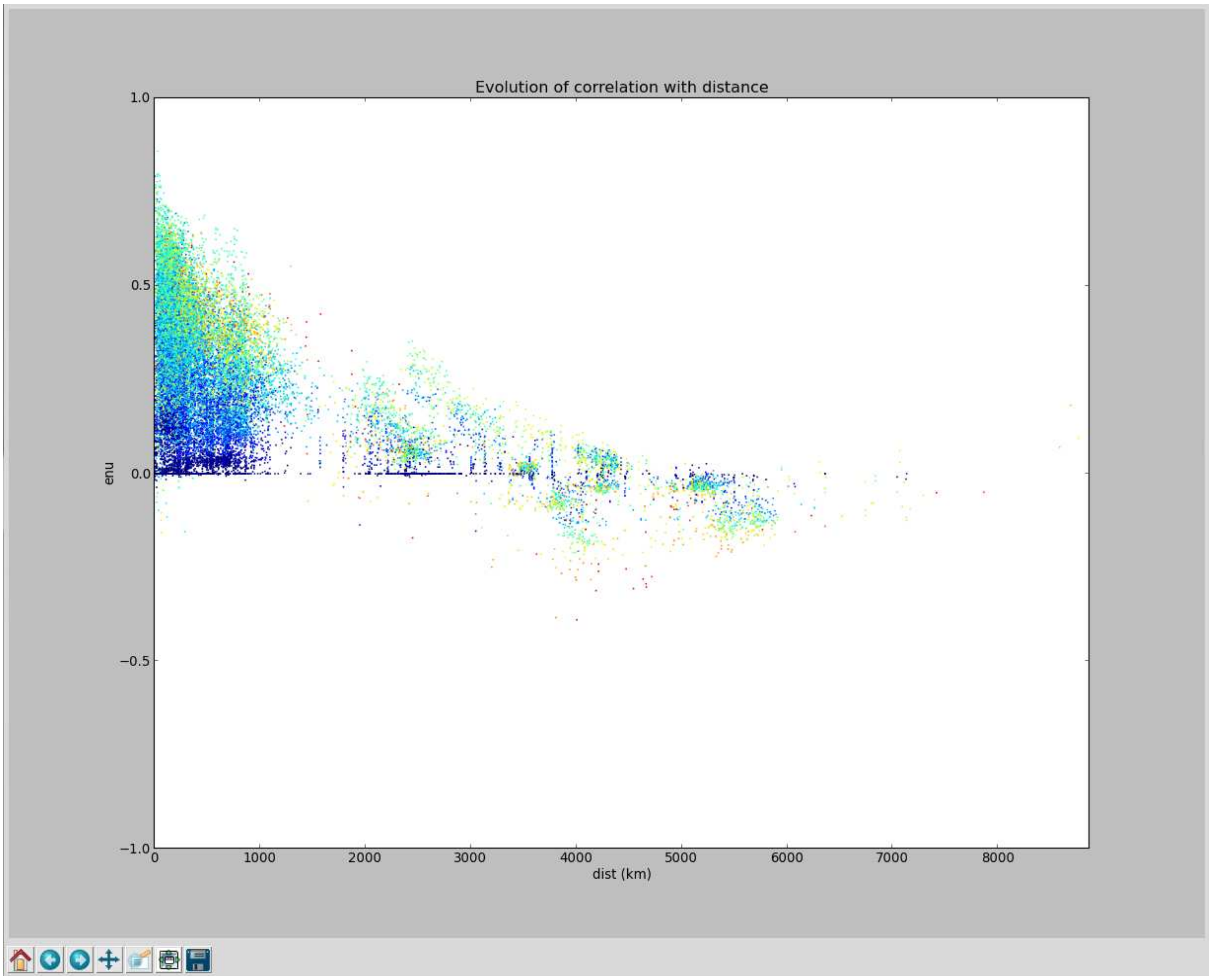


Fig 3. En x, la distance entre les stations, en y la corrélation entre leurs erreurs. Un point bleu signifie que les stations ont eu peu de jours d'observations en commun, un point rouge qu'elles en ont eu beaucoup.

Cependant, les séries temporelles ne semblent pas être beaucoup plus lisses à l'issue du lissage, ce qui est problématique. J'envisage de supprimer le splinage, ou limiter très fortement son influence, pour ne prendre en compte que les corrections calculées à partir des données de la station uniquement.

II- Détection des sauts

Pour la détection des sauts, j'utilise le module changepoint de R. Il prend en entrée les séries temporelles, et en extrait une liste de dates où des sauts ont été détectés. Une fois de plus, on procède coordonnée par coordonnée sans considérer Up, et on concatène ensuite les listes de dates.

Le module changepoint propose différents algorithmes de détection de sauts, je n'ai pour l'instant pas cherché à comprendre leur fonctionnement, je me suis contenté de tous les tester, un par un, et de sélectionner celui qui me donnait les meilleurs résultats sur plusieurs jeux de données test.

La technique actuellement utilisée est basée sur à la fois la moyenne et la variance des observations, utilise la méthode BinSeg, en appliquant des pénalités asymptotiques aux points, avec une erreur théorique de type I de 10^{-4}.

L'écueil principal rencontré lors des tests était la surdétection, parfois énorme. À l'inverse, certaines méthodes ne détectaient qu'entre 0 et 2 sauts.

Je pense que la méthode actuellement employée détecte correctement tous les sauts réels, avec un taux de faux positifs toujours non négligeable mais relativement faible.

Une des difficultés principales que j'étais censé rencontrer était la présence de séismes lents, qui se traduisent sur les séries temporelles non pas comme un saut mais comme une rampe. Cependant, ces séismes lents sont généralement détectés comme des successions rapides de petits séismes, et je suis en train d'essayer de les détecter en cherchant de telles suites de séismes. Je fais actuellement face à des bugs sur mon programme, mais je ne doute pas de pouvoir les résoudre relativement rapidement, et voir s'il est effectivement possible de les détecter de cette façon.

III- Calcul des séries temporelles finales

Maintenant les sauts détectés, il convient de les incorporer au modèle. Mon maître de stage disposait déjà d'un programme pour calculer les composantes annuelles, semi annuelles, etc., pour des séries temporelles. Ce programme permet également d'ajouter des sauts et des séismes lents, connaissant leurs dates.

Je charge donc ma série temporelle, j'ajoute d'un bloc tous les sauts, et je calcule le modèle. J'ai maintenant, pour chaque saut, sa direction et sa valeur.

On va chercher à supprimer les plus petits séismes (rappel : on a un peu de surdétection). Je dispose d'une fonction pour évaluer le niveau de bruit d'une série temporelle. Pour le moment, je me contente de procéder saut par saut, composante par composante. Si une composante du saut a une amplitude inférieure au bruit sur cette composante, elle est fixée à zéro. Si les trois composantes sont nulles, le saut est supprimé.

Maintenant que l'on a enlevé les plus petits séismes, et que l'on connaît l'amplitude des autres, on peut rechercher les séismes lents selon la méthode décrite en II- : tant que la distance entre deux sauts consécutifs est inférieure à 4 mois (durée fixée arbitrairement) et que ces sauts vont dans la même direction, ou sont jugés suffisamment faibles, on les ajoute à une liste. Une fois la liste terminée, on les remplace tous par un séisme lent, et on regarde s'il y a eu amélioration. Si oui, les sauts sont définitivement supprimés et remplacés par le séisme lent.

Au vu des résultats, il ressort que certaines suites de sauts, visuellement détectables comme des séismes lents, ne sont pas traitées par le programme, et demeurent telles quelles dans les séries finales. Je pense que cela peut être dû au fait que l'on doive remplacer tous les séismes à la suite d'un seul coup, et qu'on ne puisse pas, par exemple, garder le premier saut, puis remplacer les autres par un séisme lent. Les figures ci-dessous illustrent ce problème avec des données créées de toutes pièces.

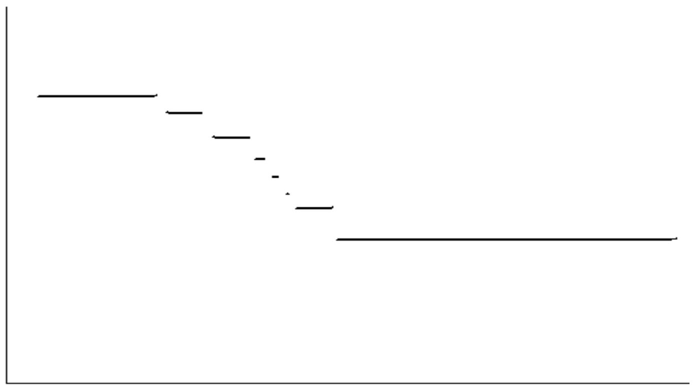


Fig 3. Les sauts tels que détectés en première approximation

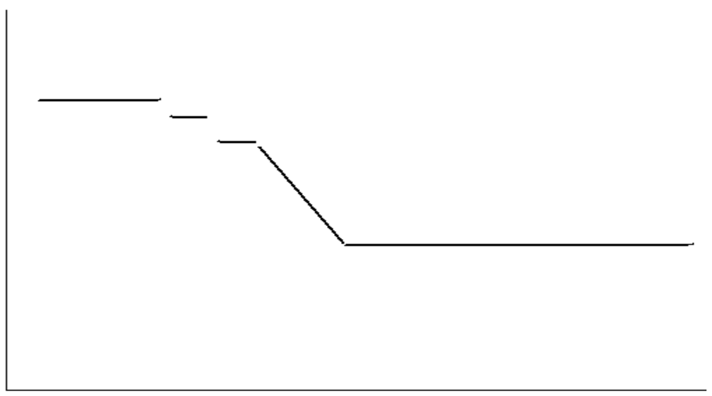


Fig 4. Le phénomène réel

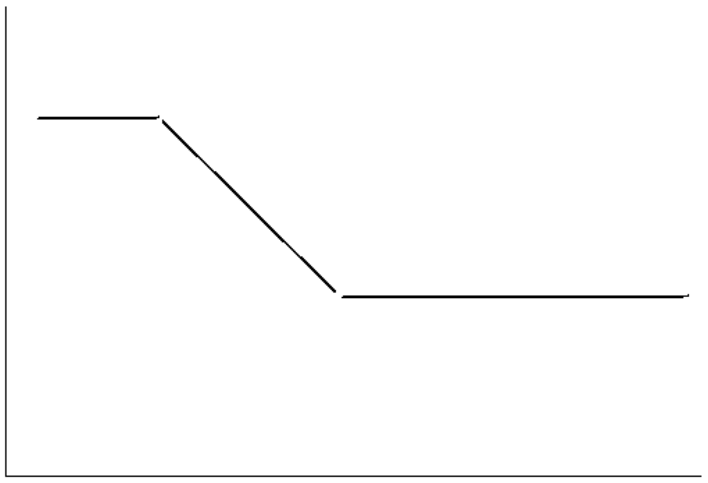


Fig 5. Ce que l'algorithme va détecter.

Il serait donc préférable de tester toutes combinaisons sauts - séismes lents séparément et de sélectionner la meilleure à la fin. Mais adapter le modèle est extrêmement long (à partir de la phase d'ajout des sauts au modèle jusqu'à la fin, le programme met déjà environ une heure), et cela ne fera que le ralentir encore plus.

D- Perspectives

La priorité est pour l'instant donnée à lisser proprement les données. Une fois les données lissées, on devrait avoir moins de séismes détectés, ce qui dans un premier temps améliorerait la qualité des résultats, mais accélèrerait aussi grandement le programme, ce qui permettrait d'améliorer ensuite la détection des séismes lents. Supprimer le splinage devrait je l'espère permettre d'atteindre ces objectifs, ça devrait être fait dans le courant de la journée je pense. En fonction des résultats, je verrai a) si les données sont effectivement plus lissées, b) si le reste de mon programme fonctionne correctement et donne de meilleurs résultats avec de meilleures données, et, si non, c) quels parties en changer.

J'ai demandé une visite du département de géodésie de l'Université Victoria de Wellington, elle devrait avoir lieu cette semaine, pour voir comment eux s'y prennent pour leurs traitements, et très certainement m'en inspirer.