# Considerations for the new dynamic\_stock\_model library

I recommend reading this post which is full of useful information:

<https://github.com/IndEcol/ODYM/issues/1>

Here is a summary of the key points and some of my own advice:

* If you are using the dynamic\_stock\_model package, you should use the one included in ODYM: this is the new version, and calculations are very fast (also to be found on BB). The old version can be extremely slow for large amounts of data. Note that the syntax of some functions are also different (there is no ExitFlag in the new version).

The way they calculate stock\_tc in the new version is probably the best: very fast, logical and concise (more in this further later):  
self.s\_c = np.einsum('c,tc->tc', self.i, self.sf)

* If you are building calculations by hand, there are several things worth considering to make the code run faster:
  + scipy.stats.norm functions are slow, so the less you use it, the better. This means that building a lifetime matrix once and for all at the beginning (eventually storing it for later use as well) and then run the model calculations is going to be faster than calling normal distribution functions in the middle of nested for loops.
  + The syntax of these functions is not very intuitive: you have different ways to use them, and some are much faster than others.  
    For example:
    - scipy.stats.norm.cdf(year, loc=mean, scale=sd) is much faster than  scipy.stats.norm(loc=mean, scale=sd).cdf(year)  
      I have been doing it wrong in the past without even noticing
    - What takes time is to call the function, then it does not matter if it runs on a single number or on a large array.   
      This means that something written like that:  
      scipy.stats.norm.cdf(range(start\_year, end\_year), loc=mean, scale=sd)  
      is going to be much faster than:

for year in range(start\_year, end\_year):

scipy.stats.norm.cdf(year, loc=mean, scale=sd)

* + Using properties of matrix algebra makes the code compact and tidy, and is usually faster (stuff like Outflow = Lifetime \* Inflow and Inflow = (Identity - Stock change)-1 . The einsum function is usually a good choice for that.
* Other things worth noting about these functions:
  + pdf, cdf and sf are calculated using some assumptions that means that they work slightly differently. All of these functions are integral of the distribution function, but they use different intervals. For example  t +/- 0.5 years for pdf, -infinity to t for cdf. In this regard, it makes more sense to use cdf or sf than pdf.
  + Especially for cases when the stdev is quite large compared to the mean lifetime, the issue is that a significant part of the distribution function occurs with t>0. There are several options to deal with this:
    - If you use sf or cdf, you will get a larger than normal outflow during the first year. There might be a justification for this (fabrication issues etc. might explain that the death rate is higher during the first year than during the second year for instance), and it makes the rest of the code easier.
    - If you use pdf and do nothing about it, you will get some remaining fraction of the cohort that will never leave the stock. Again, can be justified by heritage building, vintage collection cars, etc.
    - You can try to use a truncated normal, but I never managed to make it work perfectly. Another option is to slightly increase outflows for all years by a certain percentage so that you get rid of the whole cohort in the end.
  + In the end, these differences are marginal, and probably completely neglectable compared to the average uncertainty of our data, so it is not worth spending so much time on this. The issue is that it can be confusing for students, who would rather get a “perfect result”. These kind of details usually hinder overall comprehension, so it is good to address it in a simple way and to be able to answer questions that will arise anyway.