Calculating consumption from the PROD.nc files.

The main issue here is understanding what the units are. The unit reported in the PROD.nc file is mg N m-3 d-1.

* Is this an average per box?
* Is it per individual or total? Given the magnitude of what we calculate it seems unlikely it is per individual.

This value is reported per box (not per cell). The spatial footprints of consumption and biomass do not overlap much, which is confusing. Eat is also far more patchy than biomass.

Really the best way to do this is tracking this in the code. A search for “PROD.nc” directs us to a file named “bm->ncOPCfname” in the solution. This has a number of objects that seem related: bm->ncOpcfid, bm->ncOpcdump, and bm->ncOpcIndex. Let’s look at each of these:

* bm->ncOpcfid: this is the netcdf ID for the production/consumption output file. The most notable occurrence seems to be in the functions “writeBMphysData()” and “writeBMDiagData()” from atlantismain.c. Seems to be the same as “fid4”.
* bm->ncOpcdump: number of next output dump for growth/consumption data. Also appears in the functions above.
* bm->ncOpcIndex: File index for the growth/consumption output file.

Next, checking writeBMDiagData(). This function is defined in atdiagIO.c. The definition states:

*Routine to write the diagnostics to a netCDF file. This routine assumes that the dinfo in the MSEBoxModel is valid and corresponds with the netCDF file. Minimal checking for consistency is done here.*

This function loops over diagnostic variables stored in bm->dinfo.dtype.

bm->dinfo is a structure that contains information about diagnostic variables. It is defined in atlantisboxmodel.h. Tracers are added to it by the function “Add\_Diagnostic\_Tracer()”, which is defined in atBuildTracer.c.

Add\_Diagnostic\_Tracer() is only called from within Build\_Diagnostic\_NameList(), also in atBuildTracer.c. This function calls it a few times, one for each tracer (Graze, Prod, Eat, Grow, etc.). Here is where the “eat” tracers are added too: the code loops over functional groups; if the fg is a vertebrate, it then loops through its cohorts. sumType and dtype are 0 and 2, respectively, for the “Eat” tracers (not sure what that means). The call looks like this:

Add\_Diagnostic\_Tracer(bm, diagIndex++, name, longName, "mg N m-3 d-1", 0, 2, &FunctGroupArray[fgIndex].EatTracers[cohort], 0, FunctGroupArray[fgIndex].diagTol, 0);

Let’s track down .EatTracers. These are declared in atFunctGroup.h as an array and they seem to become part of the data structure of boxLayerInfo. It seems like the action happens in atecology.c. There are three lines where this array appears:

Once in Water\_Column\_Box():

boxLayerInfo->localDiagFlux[FunctGroupArray[guild].EatTracers[cohort]] += (double)FunctGroupArray[guild].GrazeLive[cohort];

And twice in Epibenthic\_Box(). In layers that are not the surface:

boxLayerInfo->localDiagFlux[FunctGroupArray[fgIndex].EatTracers[cohort]] += (double)FunctGroupArray[fgIndex].GrazeLive[cohort];

And in the surface layer:

boxLayerInfo->localDiagFlux[FunctGroupArray[fgIndex].EatTracers[cohort]] = (double)FunctGroupArray[fgIndex].GrazeLive[cohort];

In the two occurrences that are not the epibenthic surface layer, this tracer gets added to itself, meaning it gets summed over the water column. So, if a depth layer has 2 mg N m-3 d-1, and the one above is 3 mg N m-3 d-1, their sum becomes 5 mg N m-3 d-1. Which is a problem.

All three of these incorporate GrazeLive, defined as the Biomass of Live Prey. GrazeLive appears A LOT throughout the code. Whatever unit GrazeLive ends up having, it should be maintained in the Eat tracers NetCDF.

GrazeLive appears throughout atGroupProcesses.c, atprocess.c, and atvertprocesses().

* In atGroupProcesses.c, it appears in InvertConsumerProcesses(), Dinoflag\_Process(), and Sediment\_Epi\_Other\_Process(). It also appears in atcoral.c. All of these should refer to invertebrate consumers, so for now let’s leave them alone.
* In atprocess.c, it appears in Eat() and Invert\_Activities(), although it is always indexed by cohort.  An interesting line in Eat is:  FunctGroupArray[sp\_id].GrazeLive[cohort] = graze\_live;
* In atvertprocesses.c are the most occurrences. It appears in:
  + Vertebrate\_Activities(), where it appears in \*GrazeLive = (double)FunctGroupArray[guildcase].GrazeLive[chrt]. This last one adds the grazing for the cohort (I think) and updates \*GrazeLive. An argument to this function is &VGrazeLive, which in the function definition corresponds to \*GrazeLive. So it seems like the function modifies VGrazeLive thanks to the ampersand. Within Do-Vertebrate\_Living(), we find FunctGroupArray[guildcase].GrazeLive[cohort] = VGrazelive;.

I am finding it fairly convoluted, but the line  \*GrazeLive = (double)FunctGroupArray[guildcase].GrazeLive[chrt]; seems important. Chrt stands for cohort. In Eat(), we have the line: FunctGroupArray[sp\_id].GrazeLive[cohort] = graze\_live; So, yet another spelling of what seems to be either the same variable that then trickles down to “eat” or a step toward it. Now, in Eat() we have the lines:

/\* Add the wc and sed values \*/

                    graze\_live += spGRAZEinfo[preyID][kij][WC];

                    graze\_live += spGRAZEinfo[preyID][kij][SED];

                    graze\_live += spCATCHGRAZEinfo[preyID][kij];

So next let’s go to spGRAZEinfo. This one also shows up a ton., but in Eat() there are many occurrences like

                        spGRAZEinfo[preyID][kij][habitat] = EATINGinfo[preyID][kij][habitat] \* (double)scaled\_clear;

Which sends us to EATINGinfo (specific formula will depend on the functional response choice for the predator-prey relationship). This one also shows up loads, but one interesting call is

        prey\_eat =  prey\_avail \* pHscalar;

 EATINGinfo[preyGuildID][prey\_chrt][habitat] = (double)prey\_eat;

In Calculate\_PreyAvail(). Prey\_avail is short above that:

        prey\_avail = Get\_Gape\_Lim\_Prey(bm, llogfp, predatorGuildID, cohort, chrtstage, preyGuildID, prey\_chrt, habitat, spPREYinfo);

And it is returned by Get\_Gape\_Lim\_Prey(). In turn, for vertebrates, within that function we find:

            prey\_avail = Avail\_Fish(bm, predatorID, cohort, chrtstage, preyID, prey\_chrt, SN, VERTinfo, llogfp);

So let’s go to Avail\_Fish(). This function returns a quantity named fish\_available. These lines are very important:

/\* The amount of fish available - this value is a biomass value \*/

                    eatthis = step1 \* ((SP[prey][bpreychrt][SN\_id] + SP[prey][bpreychrt][RN\_id]) \* SP[prey][bpreychrt][DEN\_id] / (bm->cell\_vol + small\_num));

                    fish\_available += eatthis;

**The variable eatthis seems key here, and it is divided by cell volume.** The variable step1 includes bm->pSPVERTeat[guildcase][prey][chrtstage][preyage] \* pSTOCK[prey][preystock][preyage] and then a number of scalars. The first of these terms seems to be the PPREY values, or something related. Overall, step1 seems to resolve availability, and then SP contains weight of the prey (SN, RN, and DEN). So this step takes a biomass in mg N and divides it by m3, thus obtinaing mg N m-3. Following this backwards, it should go all the way to the Eat tracers. Because the model operates per time step, writing that rate out is a daily value (perhaps 12-hourly?). This seems consistent with units reported in the manual for everything related to biomass, and with many comments throughout the code that hint to a m-3 value.

So, going back to

if (it\_count == 1) {

                    if (bm->current\_layer) {

                    for (cohort = 0; cohort < FunctGroupArray[guild].numCohortsXnumGenes; cohort++) {

                            boxLayerInfo->localDiagFlux[FunctGroupArray[guild].EatTracers[cohort]] += (double)FunctGroupArray[guild].GrazeLive[cohort];

                            boxLayerInfo->localDiagFlux[FunctGroupArray[guild].GrowthTracers[cohort]] += FunctGroupArray[guild].grow[cohort][SN\_id]

                                    + FunctGroupArray[guild].grow[cohort][RN\_id];

                        }

                    }

                }

It seems like we are just adding entries of the GrazeLive array (in mg N m-3) to the localDiagFlux for a certain box and layer. The next thing to follow is boxLayerInfo->localDiagFlux. These are added to boxLayerInfo->localDiagTracer in the function Integrate\_Diag\_Variables() in atbiology.c. This function seems to have more to do with time than space, and it integrates the tracers over the adaptive time steps if necessary.

At this point I am fairly lost, but something happens before the PROD.nc file is written out because all diagnostic variables come in 2D: time and box. The only operation that appears to happen in the code is summation.

In summary, my best guess at the moment is that this variable comes out as the sum of the box\*layer values of consumption per m3. Because these are expected to change between layers, depending on how much predator, how much prey, and how much feeding occurs, we can’t just multiply those values by box volume. This will always result in an overestimate, which may be really gross if, as an extreme case, all the consumption was occurring in one shallow surface layer and was then expanded to the entire water column.

Since we don’t have layer-specific values, the next best thing is to apportion total consumption (as the sum of the rates) to individual layers based on biomass per layer (numbers \* (RN+SN), by age class). This is based on the (dangerous) assumption that consumption is only proportional to predator abundance, whereas in reality it will depend on prey availability and other factors too.