VRI Polygon to VDYP 7 Growth Curve Coercer

Written by Zachary Dirk

Lasted edited by Zachary Dirk

Document Last Generated: 12 Jul, 2017

# Table of Contents

1. [Brief Introduction](#Brief-Introduction)
   * Why Would a Polygon Need to be Coerced?
2. [How to Run the Program](#How-To-Run-the-Program)
   * What You Need
   * Other Inputs
   * Invoking The Program Via Command Line
3. [How the Program Works](#How-the-Program-Works)
   * The Algorithm
4. [Handling the Source Code](#Handling-The-Source-Code)
   * Editing the Program
   * Building the Program

# 1. Brief Introduction

The coercion tool is quite straight-forward. When given a list of VRI Polygons and an AU lookup table that maps to VDYP 7 growth curves, it will output a table of Polygons whose IDs are now mapped to the AUs, which in turn can be matched to the Growth Curves. In cases where there is no direct match from VRI Polygon to AU, it will attempt to coerce the Polygon to a best-fit Growth Curve by following an algorithm outlined further along in this document. Finally, it will also output summary information about your Polygon inventory and how it relates to the Growth Curves so that you might attempt to fix some of the problems before they arise, bypassing the coercion process.

### Why Would a Polygon Need to be Coerced?

There are a finite number of issues that can crop up and require a polygon to be coerced. This program aims to deal with as many of them as possible, based on how much input it is given.

The list of issues can be broken into two categories: issues stemming from the polygon's site index, and issues stemming from the polygon's combination of BEC zone and leading species.

##### Site Index Problems

Site index problems are the easiest problems to fix. There are two possible cases for a site index problem:

1. The polygon has no site index.
2. The polgon has a site index, but it is out of the range of all of the suitable growth curves.

The solution to problem 1. is that we set the polygon to fit the middlemost growth curve for its BEC zone/species combination. The solution to problem 2. is that we set the polygon to fit the growth curve that it is closest to.

Species and BEC zone are not numbers on a scale, so it's not necesarily possible to design a catch-all solution. Therefore we rely on the user to provide solutions to the problems that arise, and if the user provides no solutions we merely report the error and pass over it. This is to avoid making incorrect assumptions.

##### BEC Zone/Species Combination Problems

BEC zone/species combination problems are a little bit more complex because they require changing either the species or the BEC zone of the polygon in order to coerce them. Again the problems tend to fall into two big categories:

1. The species has no growth curves for this BEC, or for any BEC.

The solution to both problems is to treat the species as though it were a different one. To do so we use a map provided by the user in the form of a two column csv. Column 1 is the species, and column 2 is the substitute species. Examples of both these extra input documents will be given in section 2.

# 2. How to Run the Program

The program is designed to be run from the command line, so that it may be used in scripting processes.

### What You Need

For input, the bare minimum that is required is a comma delimited (csv) file containing the Polygon information, and a second csv file containing the Growth Curve information. None of the cells within the file may contain commas, as this will interfere with the programs ability to parse the information.

##### Polygon Input Table

The polygon table must include four specific fields: ID, site index, BEC zone, and leading species. The names of the fields must be specifically as follows: ID, SiteIndex, BEC, LdSpp, but they can appear in any order, and are not case-sensitive. Example table:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | SiteIndex | BEC | LdSpp |
| 1 | 16.2 | IDF | Lodgepole Pine |
| 311 | 12 | SWB | Douglas-fir |
| 19439 |  | IMA | Subalpine Fir |

##### AU Lookup (Growth Curve) Input Table

The growth curve table follows similar specifications, it must contain five fields named thusly: AU, SI\_MIN, SI\_MAX, BEC, LdSpp, where SI\_MIN is the minimum bound for site index and SI\_MAX is the maximum bound for site index. Again, these field names are specific, but they are not case sensitive. Example table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AU | SI\_MIN | SI\_MAX | BEC | LdSpp |
| 1 | 5 | 35 | BAFA | Aspen |
| 36 | 15 | 25 | BG | Ponderosa pine |
| 218 | 5 | 25 | BWBS | Western white pine |

With these two files, the program will be able to match all the polygons that do not require any coercion, as well as coercing any polygons whose issues lie in the site index. It will not, however, be able to match any polygons whose issues lie in their BEC/Species combination. There are two optional inputs that will deal with that.

### Other Inputs

##### Species Substitution Input Table

The optional input is a species substitution table. Again in the form of a csv, this table dictates how to treat species that do not have any growth curves. It contains two columns: species, and substitute. When the program can't find a growth curve for a certain polgyon, it will check if that polygon's species is in the "species" column here and if so it will replace the species with the corresponding species in the "substitute column". Example table:

|  |  |  |
| --- | --- | --- |
| Species | Bec | SubstituteSpecies |
| Sitka Alder | SBS | Hardwoods |
| Red Pine | ICH | Lodgepole pine |
| Subalpine Larch | BWBS | Alpine Larch |

##### Output Directory

The final optional input is the output directory. This optional input dictates where the program will write its log files, summary files, and output csv. Note: the output directory must exist BEFORE you invoke the program. The program will not create the directory for you. If you do not specify an output directory, it will use the output directory found at the same level as the src directory.

### Invoking the Program Via Command Line

The program is invoked by running its .exe file. However because it requires so many inputs, this must be done via the command line in order to provide the arguments. The program does not read the arguments in any specific order because some arguments are optional. Instead, you must provide flags that denote which argument is next. Here is the list of flags and what they denote (note: they are not case sensitive):

-AU Coming next is the AU lookup table (the growth curves)

-P Coming next is the polygon inventory

-SUB Coming next is the species substitution table

-O Coming next is the output directory

-Q This flag is not for input, rather it denotes that the user wants quiet mode: using the flag will silence all terminal output (except for errors). It can be followed by nothing or by another flag.

Five directories can be a lot to type out, especially if the program is going to be invoked multiple times with different parameters. The best solution is to create some sort of batch file that will contain your invokation of the program. This would also all you to script the process by calling the batch file. Example invocations:

MatchGrowthCurves.exe -AU C:/users/zdirk/gycurves/inputs/AULut.csv -p C:/users/zdirk/gycurves/inputs/polygons.csv -q -SUB C:/users/zdirk/gycurves/supporting/sub.csv

MatchGrowthCurves.exe -p M:\Spatially\_explicit\01\_Projects\13\_PICS\03\_BC\_GCBMruns\05\_working\01\_inventory\01\_VRI\_processing\input\polygons.csv -AU M:\Spatially\_explicit\01\_Projects\13\_PICS\03\_BC\_GCBMruns\05\_working\01\_inventory\01\_VRI\_processing\input\AU\_LUT\_fromSIT\_reformatted.csv

# 3. How the Program Works

### The Algorithm

The program is very much just a script that applies an algorithm to every polygon in the inventory, so the simplest way to explain what the program is doing is just to show the algorithm (scoping is done by indentation similar to python):

For every polygon in the inventory, do:  
 try and match the polygon to a curve  
 if the match was successful, do:  
 skip to the next polygon  
   
 check to see if there are any growth curves for this polygons combination of BEC and Species  
 if there are none, do:  
 check to see if the user provided a species substitution map  
 if they did, do:  
 try and change the species  
 if you were able to change the species, do:  
 try and match the polygon to a growth curve again  
 if you were successful, do:  
 skip to the next polygon  
   
 check to see if there are any growth curves for this polygon's possible new combination of BEC and Species  
 if there are, do:  
 check if the polygon has a site index  
 if it does have a site index, do:  
 coerce the site index to the nearest possible growth curve for this combination of BEC and species  
 otherwise, do:  
 coerce it to the middlemost growth curve for this combination of BEC and species

Once the algorithm has completed, there are two possibilities:

1. The polygon found a match
2. The polygon did not find a match

In case 1, that's great, that's exactly what we wanted. In case 2, the most likely reason that the polygon did not find a match was because either the species substitution table wasn't provided, or it was incomplete. To match every polygon, the best solution is to complete the missing optional inputs and rerun the program.

# 4. Handling the Source Code

The source code for the program was written in C++ as a learning experience. The source code is distributed with the working copy on the M drive. A remote repository exists at R:/PICS/Git/GYCurveMatcher.git.

### How to Edit the Program

To edit the program, you'll need a copy of the code. If you don't have it, install Git, and then clone the remote repository to your local drive. The program is entirely self-sufficient, it has no code dependencies beyond the standard C++ library. So, now that you've got all the code, you can make your changes as you see fit.

Once you're finished editting the code, if you'd like to update the working network copy then it's a two step process:

First use the git push command in your local repository to push the code updates back to the remote repository. Make sure you build the project locally before you push it back to the remote repository, or the .exe will not be updated! If you don't know how to build a project, see the next step.

Second, use the git pull command in the local repository on the M drive to pull the code updates to the M drive. If you built the project, then it will update the existing .exe on the network drive and the new program will be in place.

### How to Build the Program

(This assumes you are working on a windows machine)

To build any C++ program, you need several tools, including a compiler, a linker, and the standard C++ library. Microsoft offers these for free in their Visual C++ package, which is what was used to create this program. If you prefer an IDE, Visual Studio Community would be your choice. If you prefer command line building, then just download the Build Tools. Microsoft has tutorials to help with installation, as you will have to add the compiler path, the lib paths, and the include paths to your environment variables.

This program specifically was writting using a text editor and the command line. The toolchain used was the Microsoft Visual Studio Build Tools, which uses cl.exe for compiling and linking and comes with the standard library, as well as nmake.

Because this program was written without the IDE, a makefile was necssary. If you've never seen a makefile before, Microsoft does also have some documentation about them that might explain how to use, and edit it. If you chose the same toolchain, you may be able to use the makefile without modificiations. Otherwise, you should be able to build the program by modifying the parameters at the top of the makefile to suit your environment.

In general, unless you're running a debugger, you should build this project with full optimization. It does not add much compile time, but it saves a lot of running time.

If you make it through the cleanbuild process without compiler errors and it produces a .exe file, then it's likely that it was a successful build (but not necessarily succesful logically!). At this point you can follow the [instructions](#How-To-Run-the-Program) on how to run the program, to run your newer program.