Report on use of the GFS FV3 GRIB2 datasets obtained from ftp.ncep.noaa.gov

23 July 2019

Don Morton
Boreal Scientific Computing
Fairbanks, Alaska, USA
Don.Morton@borealscicomp.com

Executive Summary

NCEP replaced its GFS model with the GFS FV3 model in June 2019, and the resulting GRIB2 files made available through ftp.ncep.noaa.gov have some differences in structure, beyond the expected differences in values of a new NWP model. This document outlines my exploration of these differences as they apply to FLEXPART usage.

The short story is that the structural differences in the GRIB2 met files do result in differences in the FLEXPART 3d data, due to the way that FLEXPART code reads this data in. Experiments have been performed, comparing FLEXPART runs driven by the FV3 files available from NCEP, and from modified FV3 files in which the relevant structural differences are removed. These differences in input data show up at the 1 to 150 mb levels in the *u, v, w, qv, t* and *height* variables processed by *readwind_gfs()*. The experiments I performed had a single OUTGRID layer with a top of 150 meters, and five-day backward and forward simulations produced SRS files that were unix diff identical - zero differences between the FV3- and modified-FV3-driven simulations.

It's clear to me that the new FV3 files, when ingested by FLEXPART, put incorrect data in the temperature arrays at the 15 and 40 mb levels, and this results in differences at the high-altitude levels for other variables. It's not clear to me whether these differences should be a concern to anybody, but I am writing this report "just in case." If this is a concern, there are two possible fixes I can think of

- Modify the FLEXPART code so that it will read the levels correctly. This would be a little involved, and error-prone
- Modify the FV3 files with the *eccodes grib_copy* utility to "cut out" the offending layers. This is easy

---

Background

In June 2019 NOAA replaced the core of its Global Forecast System (GFS) with a Finite-Volume Cubed-Sphere (FV3) - [NOAA upgrades the U.S. global weather forecast model](#).

An ongoing comparison of the two is available at - https://www.emc.ncep.noaa.gov/mmb/gmanikin/fv3gfs/fv3images.html. This was last accessed on 23 July 2019, and I imagine it will vanish in the near future once the original GFS runs are halted (Autumn 2019?).

The FV3 GRIB files can be downloaded from (for example)

ftp://ftp.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gfs.20190625/18/

First, note that the directory structure is slightly different than it was pre-FV3.

Of more interest is that the new GRIB files have a couple of oddities at the 15mb and 40mb pressure levels. In the following, I've excerpted three columns from the *grib_ls* output on one of these files. What's notable is that the new FV3 GRIB files have data at the 15 mb and 40 mb pressure levels - these levels were not present in the old GFS. A little more worrisome is that these new pressure levels don't have the *u*, *v*, and *r* values that are present in the other pressure levels.

```
.
.
.
isobaricInhPa  10         gh
isobaricInhPa  10         t
isobaricInhPa  10         r
isobaricInhPa  10         u
isobaricInhPa  10         v
isobaricInhPa  10         absv
isobaricInhPa  10         o3mr
```

```
isobaricInhPa  15          gh
isobaricInhPa  15          t
isobaricInhPa  15          absv
isobaricInhPa  15          o3mr
isobaricInhPa  20          gh
isobaricInhPa  20          t
isobaricInhPa  20          r
isobaricInhPa  20          u
isobaricInhPa  20          v
isobaricInhPa  20          absv
isobaricInhPa  20          o3mr
isobaricInhPa  30          gh
isobaricInhPa  30          t
isobaricInhPa  30          r
isobaricInhPa  30          u
isobaricInhPa  30          v
isobaricInhPa  30          absv
isobaricInhPa  30          o3mr
isobaricInhPa  40          gh
isobaricInhPa  40          t
isobaricInhPa  40          absv
isobaricInhPa  40          o3mr
isobaricInhPa  50          gh
isobaricInhPa  50          t
isobaricInhPa  50          r
isobaricInhPa  50          tcc
isobaricInhPa  50          u
isobaricInhPa  50          v
isobaricInhPa  50          absv
isobaricInhPa  50          clwm
isobaricInhPa  50          icmr
isobaricInhPa  50          rwmr
isobaricInhPa  50          snmr
isobaricInhPa  50          grle
isobaricInhPa  50          o3mr
.
.
.
```

This was causing problems in a utility designed to check GRIB files before they are used, because the utility would try to assure that any of the primary variables like *u*, *v*, and *t* were available on the same pressure levels. In the case of the new FV3 files, the *t* values were available on the 15 and 40 mb pressure levels, but the other variables were not.

We had already tested FLEXPART simulations with the new FV3 met files and noted that they ran to completion, and that differences relative to the previous GFS met files “seemed” to be within the bounds that we might expect for two different NWP models. But, we began to wonder if the introduction of *t* at two additional pressure levels might introduce errors into the FLEXPART code. We found that they did.

Analysis of code, leading to hypothesis

I analyzed the code from FPv9.3.2 - a version used solely by CTBTO. I have since looked at the analogous code in FPv10.3, and it looks to me like the same problems will manifest themselves there.

This is a preliminary explanation - based on reading the source code - of how the NCEP pressure levels are processed

- In *gridcheck_gfs.F90*, we read a single GRIB file to figure out how many levels there are, etc. As each message is read, if it is a *UU* (wind) message, then we store its level (in millibars) in array *pres()*.   Through several steps, these pressure levels make their way into array *akm()*. So, when this routine is complete, the global array *akz()* contains a sorted list of the pressure levels for UU, and it's assumed that this will correspond to the same pressure levels for *v*, *r*, *w* and *t*. Sabine has a comment in here which says that we assume the pressure levels are in descending order, and has code to make sure this happens, but it looks like this will only work if the array does not start with a mixed order.

- Then, *readwind_gfs.F90* is used to read individual GRIB files, using some of the variables (like array *akz()*) that were created in *gridcheck_gfs.F90* to help with later processing.

    - Before reading the messages, variables like *numpu* (number of *u* pressure levels read so far), *numpt* (number of *t* pressure levels read so far), etc. are set to zero.
    - As each message is read, if it corresponds to *u*, the pressure level for the message is read, and then the array *akz()* is scanned to find the index that corresponds to that pressure level. Then, the full 2D slice is read into the 3D array *uuh(:,:,numpu)*. If all goes well, then after all messages have been read, uuh should have 2D horizontal slices for all pressure levels. This also applies to

the other 3D variables, *v*, *r*, *t* and *w* (*w* is a little more complicated, because it's available only at lower atmospheric levels, but it works).

So let's consider the reading in of *t*, available at two extra pressure levels (15 mb and 40 mb), based on an *akz()* that was created from *u*, and doesn't have those two extra pressure levels... When it is determined that the message contains variable *t,* the code searches for the pressure level of this message (variable *current_grib_level*) in the array of pressure levels, *akz()*:

```
do ii=1,nuvz
    if (current_grib_level .eq. akz(ii)) numpt=ii
end do
```

In the case of a 15 mb or 40 mb pressure level, the `if` statement will never evaluate to `True`, so *numpt* will retain the value from the last message that contained *t*. This means that the current 2D horizontal slice will overwrite the one from the last message. This will happen twice - once when reading the 15 mb message, and again when reading the 40 mb message. The rest of the code will assume these are all correct, and when it "thinks" it's using 10 mb temperatures, it will really be using 15 mb temperatures, and when it "thinks" it's using 30 mb temperatures, it will really be using 40 mb temperatures.

40 mb is approximately 22km altitude, and 15 mb is approximately 28 km altitude. My gut feeling is that the effects of this error are negligible.

So, the hypothesis I come to is that FLEXPART (versions 9.3.2, 10.3, and many others), upon reading FV3 met files, will incorrectly store 15 mb temperatures in the 10 mb level of the array, and 40 mb temperatures in the 30 mb level of the array.

Testing the hypothesis that two FV3 pressure levels are ingested incorrectly by FLEXPART

The primary experiment I set up consisted of two identical FLEXPART simulations, varying only in the structure of the FV3 met file ingested. One simulation used the original FV3 met files and the other simulation used modified FV3 files. These files were modified as follows:

```
$ grib_copy -w level!=15,level!=40 old.gr2 new.gr2
```

In addition to removing the four 15 mb messages and the four 40 mb messages, this also removes four 40 Pa messages (which are not used by FLEXPART) and two 40-meter u,v messages (also note used by FLEXPART), for a total of fourteen messages removed.

In the FPv9.3.2, getfields.F90, the following code was added in the declarations:

```
#ifdef FV3DB
    INTEGER fv3db_levelnum
    INTEGER fv3db_idx
#endif
```

and the following was added just after the call to readwind_gfs():

```
             call readwind_gfs(indj,memind(1),uuh,vvh,wwh)

#ifdef FV3DB
PRINT *, 'FV3 Debugging Output...'
PRINT *, 'indj, memind(1): ', indj, memind(1)
PRINT *, ' '

PRINT *, '        level    press level    TT_MAX             TT_MIN             TT_AVE'
DO fv3db_levelnum=18,31
    PRINT *, fv3db_levelnum, akz(fv3db_levelnum), &
&            MAXVAL(tth(:,:,fv3db_levelnum,1)), &
&            MINVAL(tth(:,:,fv3db_levelnum,1)), &
&            SUM(tth(:,:,fv3db_levelnum,1)) / SIZE(tth(:,:,fv3db_levelnum,1))
ENDDO

PRINT *, ' '

PRINT *, '        level    press level    UU_MAX             UU_MIN             UU_AVE'
DO fv3db_levelnum=18,31
    PRINT *, fv3db_levelnum, akz(fv3db_levelnum), &
&            MAXVAL(uuh(:,:,fv3db_levelnum)), &
&            MINVAL(uuh(:,:,fv3db_levelnum)), &
&            SUM(uuh(:,:,fv3db_levelnum)) / SIZE(uuh(:,:,fv3db_levelnum))
ENDDO

PRINT *, ' '

PRINT *, '        level    press level    VV_MAX             VV_MIN             VV_AVE'
DO fv3db_levelnum=18,31
    PRINT *, fv3db_levelnum, akz(fv3db_levelnum), &
```

```
&             MAXVAL(vvh(:,:,fv3db_levelnum)), &
&             MINVAL(vvh(:,:,fv3db_levelnum)), &
&             SUM(vvh(:,:,fv3db_levelnum)) / SIZE(vvh(:,:,fv3db_levelnum))
ENDDO

STOP
#endif
```

This had the effect of dumping key statistics (max, min, average) for levels 18 through 31 of the *tth*, *uuh* and *vvh* arrays.   If the hypothesis was correct, I would expect that the statistics produced by ingesting both met files would be identical, EXCEPT for the 10 mb and 30 mb pressure levels (levels 24 and 26 in the arrays) in the *tth* variable. The *uuh* and *vvh* variables should be identical. This was, indeed, the case, as seen below.

Original FV3 Met File

level	press level	TT_MAX	TT_MIN	TT_AVE
18	25000.0000	240.699997	202.100006	223.488281
19	20000.0000	237.704697	197.704697	217.862366
20	15000.0000	234.846863	195.046860	213.431168
21	10000.0000	231.209885	189.509888	208.677231
22	7000.00000	230.100006	190.100006	208.705551
23	5000.00000	230.373566	185.173553	211.106491
24	3000.00000	231.099655	180.599655	212.623291
25	2000.00000	236.713669	173.513672	217.807236
26	1000.00000	239.597244	176.197235	219.870667
27	700.000000	250.109833	179.809830	228.384338
28	500.000000	257.320374	174.620361	234.090302
29	300.000000	268.524231	194.624222	244.970078
30	200.000000	279.890259	202.390274	253.036102
31	100.000000	283.813904	208.713913	257.377411

level	press level	UU_MAX	UU_MIN	UU_AVE
18	25000.0000	82.7371521	-45.5628471	11.4414339
19	20000.0000	80.0236816	-30.3763218	12.7553673
20	15000.0000	69.9404678	-33.8095284	12.8670254
21	10000.0000	63.9847946	-36.9052048	10.1393251
22	7000.00000	58.2364883	-23.4335098	8.46585846
23	5000.00000	61.7349739	-18.0050240	7.80478239
24	3000.00000	70.4412537	-23.9187450	7.26333427
25	2000.00000	80.9172745	-27.1127224	6.78807354
26	1000.00000	110.139793	-35.8602066	4.63955450
27	700.000000	125.496750	-40.9632454	4.92044544
28	500.000000	133.159744	-45.5002480	6.90335608
29	300.000000	135.399704	-49.7002945	10.5503769
30	200.000000	133.309418	-51.1105843	11.9342213
31	100.000000	148.408508	-52.1314926	13.4070730

Modified FV

level	press level	TT_MAX	TT_MIN	TT_AVE
18	25000.0000	240.699997	202.100006	223.488281
19	20000.0000	237.704697	197.704697	217.862366
20	15000.0000	234.846863	195.046860	213.431168
21	10000.0000	231.209885	189.509888	208.677231
22	7000.00000	230.100006	190.100006	208.705551
23	5000.00000	230.373566	185.173553	211.106491
24	3000.00000	232.539719	175.539719	214.837738
25	2000.00000	236.713669	173.513672	217.807236
26	1000.00000	245.136002	176.636002	223.705826
27	700.000000	250.109833	179.809830	228.384338
28	500.000000	257.320374	174.620361	234.090302
29	300.000000	268.524231	194.624222	244.970078
30	200.000000	279.890259	202.390274	253.036102
31	100.000000	283.813904	208.713913	257.377411

level	press level	UU_MAX	UU_MIN	UU_AVE
18	25000.0000	82.7371521	-45.5628471	11.4414339
19	20000.0000	80.0236816	-30.3763218	12.7553673
20	15000.0000	69.9404678	-33.8095284	12.8670254
21	10000.0000	63.9847946	-36.9052048	10.1393251
22	7000.00000	58.2364883	-23.4335098	8.46585846
23	5000.00000	61.7349739	-18.0050240	7.80478239
24	3000.00000	70.4412537	-23.9187450	7.26333427
25	2000.00000	80.9172745	-27.1127224	6.78807354
26	1000.00000	110.139793	-35.8602066	4.63955450
27	700.000000	125.496750	-40.9632454	4.92044544
28	500.000000	133.159744	-45.5002480	6.90335608
29	300.000000	135.399704	-49.7002945	10.5503769
30	200.000000	133.309418	-51.1105843	11.9342213
31	100.000000	148.408508	-52.1314926	13.4070730

level	press level	VV_MAX	VV_MIN	VV_AVE
18	25000.0000	60.1046066	-59.1953926	4.61703129E-02
19	20000.0000	60.5235062	-63.8764954	-0.141465545
20	15000.0000	42.1624603	-45.8475380	-0.421888620
21	10000.0000	31.0849590	-24.7650414	-4.07871343E-02
22	7000.00000	40.0117645	-24.3782349	-9.11320299E-02
23	5000.00000	42.1160011	-19.6340008	-0.266243249
24	3000.00000	49.8426819	-19.3073196	0.132733643
25	2000.00000	55.2268600	-22.0731392	9.13939402E-02
26	1000.00000	53.2846832	-24.3253155	0.151836589
27	700.000000	51.4220848	-31.9379158	-8.45138505E-02
28	500.000000	49.1979446	-36.5120544	-9.76039618E-02
29	300.000000	52.5225945	-35.6674042	0.189863190
30	200.000000	38.0044250	-34.1555748	0.342470437
31	100.000000	38.1457481	-34.2642517	8.36446658E-02

level	press level	VV_MAX	VV_MIN	VV_AVE
18	25000.0000	60.1046066	-59.1953926	4.61703129E-02
19	20000.0000	60.5235062	-63.8764954	-0.141465545
20	15000.0000	42.1624603	-45.8475380	-0.421888620
21	10000.0000	31.0849590	-24.7650414	-4.07871343E-02
22	7000.00000	40.0117645	-24.3782349	-9.11320299E-02
23	5000.00000	42.1160011	-19.6340008	-0.266243249
24	3000.00000	49.8426819	-19.3073196	0.132733643
25	2000.00000	55.2268600	-22.0731392	9.13939402E-02
26	1000.00000	53.2846832	-24.3253155	0.151836589
27	700.000000	51.4220848	-31.9379158	-8.45138505E-02
28	500.000000	49.1979446	-36.5120544	-9.76039618E-02
29	300.000000	52.5225945	-35.6674042	0.189863190
30	200.000000	38.0044250	-34.1555748	0.342470437
31	100.000000	38.1457481	-34.2642517	8.36446658E-02

Further, because the grib messages contain max, min and average values, I should be able to process the files with the *eccodes grib_ls* and compare those values with the ones obtained above and find that, in the original FV3 met file, the *t* values at 15 mb end up in the 10 mb level of the *tth* array, and the *t* values at 40 mb end up in the 30 mb level of the *tth* array:

```
$ grib_ls -p shortName,level,maximum,minimum,average -w typeOfLevel=isobaricInhPa GX19061300
shortName    level       maximum     minimum     average
gh           1           50031.4     40964.7     47001.4
t            1           283.814     208.714     257.314
r            1           0.1         0           6.27116e-05
u            1           148.409     -52.1315    13.4074
v            1           38.1457     -34.2643    0.0777974
o3mr         1           8.30899e-06  3.38339e-06  5.40342e-06
.
.
.
t            15          239.597     176.197     219.9
.
.
.
t            40          231.1       180.6       212.72
.
.
.
```

Again, this is exactly the case - the max and min values agree according to the hypothesis. None of the average values agree, but the FLEXPART array sizes are larger than the number of values in a GRIB message, so one would not expect the average values to agree.

So, to summarize, the evidence above strongly supports the hypothesis that **FLEXPART (versions 9.3.2, 10.3, and many others), upon reading FV3 met files, will incorrectly store 15 mb temperatures in the 10 mb level of the array, and 40 mb temperatures in the 30 mb level of the array.**

Other experiments performed

Two additional experiments were performed. They were actually performed before the decisive one described above, but weren't conclusive enough. Still, they add valuable insight.

I - Comparison of FLEXPART output driven by original FV3 versus modified FV3 met files

A FLEXPART configuration was defined with a single 12-hour release over Quito, Ecuador, for runs of 120 hours. The OUTGRID had a single level defined at 150 meters, and the raw FLEXPART output was converted to an SRS file.

A forward and a backward set of simulations were performed, each driven by both original FV3 met files and modified (15 mb and 40 mb levels removed) FV3 met files. The resulting SRS files were compared with the Unix *diff* utility, which reported exactly identical outputs for the forward case, and the same for the backward case. This was interesting, but not a complete surprise, given that the altitude of 30 mb is approximately 25,000 meters!

This was only one experiment, but it gave me the feeling that at least at low altitudes, the effects of the problem described above are negligible, if not absent.

II - Comparison of GRIB2FLEXPART output produced from original FV3 versus modified FV3 met files

With the *grib2flexpart* utility available in FLEXPART v9.3.2, we are able to read GRIB files, perform the processing that FLEXPART normally performs on these files, and then write to NetCDF4 (NC4) files for later use. With this output I thought it would be instructive to compare the processed fields produced from both original FV3 and modified (15 mb and 40 mb levels removed) FV3 met files. What I found was that all of the 3D fields exhibited differences (relative to the original and modified FV3 inputs) from approximately levels 24 to 31, which corresponds to those levels at 30 mb and above. I don't understand all the details of the processing, but I think my findings can be summarized simply by showing the two *height* arrays created through the processing. The first one comes from the original FV3 input, the second from the modified FV3 input:

```
height = 0, 188.7394, 380.9402, 578.2631, 781.939, 1213.639, 1678.051,
    2175.971, 2709.85, 3279.624, 3888.484, 4540.39, 5245.276, 6012.5,
    6848.926, 7769.207, 8795.345, 9957.576, 11304.94, 12985.91, 15334.57,
    17387.61, 19312.11, 22184.87, 24422.83, 28216.06, 30192.36, 32094.13,
    35085.36, 37566.02, 41934.45, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ;

height = 0, 188.7394, 380.9402, 578.2631, 781.939, 1213.639, 1678.051,
    2175.971, 2709.85, 3279.624, 3888.484, 4540.39, 5245.276, 6012.5,
    6848.926, 7769.207, 8795.345, 9957.576, 11304.94, 12985.91, 15334.57,
    17387.61, 19312.11, 22161.31, 24380.83, 28208.81, 30203.09, 32104.87,
    35096.09, 37576.75, 41945.19, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ;
```

One finding that surprised me was that some of the pairwise differences were large. Since these were netCDF files I was able to write a Python-numpy program that found the pairwise differences of a level generated by original and modified FV3 inputs, and then find the maximum absolute difference - in one case I was finding differences of up to 21 Kelvins at an arbitrary point. I didn't pursue this any further, but it dawned on me that "if" someone is using FLEXPART (or *grib2nc4*) at these high altitudes, maybe they have something to be concerned about - I really don't know.

Summary

There is no doubt that use of the new FV3 files will result in FLEXPART writing 15 mb temperature values into its 10 mb level in *tth*, and writing the 40 mb temperature values into its 30 mb level. After normal processing of the 3D variables, these errors seem to expand to other levels, but, in my limited testing, it seems like the effects are at the 30 mb level and above.

So, this is clearly a “bug,” but at least for low-altitude simulations it doesn’t strike me as one that’s worthy of additional work. One could refactor the code in readwind_gfs() (if they did, they might also want to get rid of the assumption that pressure levels in the GRIB file are ordered), but this would be a little tedious and prone to introducing more errors.

It seems to me that an easier fix - if people found it necessary - would be to use *eccodes grib_copy* to remove the offending levels.