Using nonlinear *p*-multigrid in HORSES3D

INTRO - WHY USE NONLINEAR *P*-MULTIGRID?

Nonlinear *p*-multigrid, also known as Full Approximate Scheme (FAS) drastically improves CPU time needed to converge steady-state cases in comparison to the default explicit solver. In extreme examples, the performance can increase 100x, so it's a difference of getting the results after the coffee break and the next day. Following sections explain the flags that are used in FAS.

USING EXPLICIT FAS WITH HORSES3D

The 2 most basic flags in the control file are (just to use FAS):

```
time integration = fas ! this flag specifies that we want to use FAS as a solver

simulation type = steady-state ! this flag specifies that we solve a STEADY-STATE case
```

It is also possible to use FAS for unsteady simulation using Dual Time Stepping to converge a local problem, but this is not covered in here. Then, there are also few necessary flags that must be explicitly specified in order for the solver to actually work:

```
multigrid levels = INTEGER ! this flag specifies total number of levels used in the FAS. E.g. 1 means
that there is only one level, so essentially no coarse grid acceleration is provided. If we solve a case
with P=4 and default delta N (see next box), setting this number to 4 means we use 4 levels (3
additional coarse levels for P3,P2,P1), i.e. the full potential of the FAS is used.

convergence tolerance = REAL ! this flag specifies the tolerance, to which the case will be converged

cfl = REAL ! this flag specifies the CFL number
```

Note that instead of the *cfl* flag, we can use *dt*. However, this is not advised since the solver will use the same dt on all levels and we want to have a higher time step on coarse levels. Next, there are several optional flags for different parameters.

delta N = INTEGER ! this flag specifies how the polynomial order changes on the next, coarse level, i.e. $P_{coarse} = P_{fine} - \Delta N$. The default is 1. If we have P3 case and set it to 2, then the next coarse level has P1.

dcfl = REAL ! this flag specifies the the visous CFL number. Default is the same as cfl, which has to be specified.

mg sweeps = INTEGER ! this flag specifies number of sweeps on each level. The default is one. Note that there are several different ways of specifying number of sweeps in FAS, check documentation for more detailed explanation. My personal favourite is mg sweeps exact = [INTEGER, INTEGER, …, INTEGER], where you can explicitly specify number of sweeps on each level.

Defining the flags above is sufficient to use FAS as a solver. Next, there are several **OPTIONAL** flags that can improve performance.

CFL RAMPING

Let's start with the CFL ramping/boosting strategy and a few words. It is a common thing for a simulation to crash with a high CFL, but successfully run with lower CFL. However, this is not constant for the whole simulation. It often happens that the restriction is only for initial few hundred iterations until the flow develops and residual drops few orders of magnitude. Instead of saving the current solution and restarting it with higher CFL, we can start with low CFL and systematically increase it during a run.

Important thing to say here is that there are several scientific approaches on how to increase CFL during the simulation for the best performance, see for instance Hoshyari et al. *Efficient Steady-State Convergence for a Higher-Order Unstructured Finite Volume Solver for Compressible Flows*. They are not hard to implement, however, they have to be carefully tested on several different test cases. Since I couldn't figure out quick enough what would be the best, automatic strategy and had to optimize each test case separately, I decided to put something straight forward.

cfl boost = STRING ! this flag specifies the option for cfl boosting. It's either 'linear' or 'exponential'. The former is simply $CFL = CFL + \Delta CFL \cdot CFL$ and the latter is $CFL = \Delta CFL \cdot CFL$.

dcfl boost =STRING ! the same as above, but for DCFL

cfl boost rate = REAL ! this is ΔCFL from the flag above. Default 0.1.

cfl max = REAL! Cfl will only increase up to the value specified by this flag. Default 1.0.

dcfl max = REAL! As described above for DCFL. Default 1.0.

INITIALIZATION

The initialization strategy is another way of changing the FAS parameters during the run, such that at the beginning we use something that is slower and more robust, but once the flow is developed, we use a faster solver. The CFL ramping strategy described above is a gradual change, where we use the same solver throughout the run. If we use the *initialization*, the parameters change suddenly at the pre-defined moment. The idea here is that we specify ALL the flags for FAS for a desired solver (e.g. semi-implicit solver), however, at the beginning, FAS uses ERK5 smoother with the 'initial' options (described below) until the residual is dropped sufficiently, and only then, FAS starts to use the solver we have defined in the control file.

```
mg initialization = LOGICAL ! set .true. to use it. If true, FAS will use ERK5 smoother until the residual drops to the value defined by initial residual.

initial residual = REAL! this is how we define the moment, where the initialization part stops. Until here, we use ERK5 smoother. After this threshold is reached, the solver defined by mg smoother flag is used with corresponding parameters. Default 1.0.

initial cfl = REAL! Initial cfl for ERK5 smoother. Same for DCFL (for simplicity). Default 0.5.

initial preconditioner = STRING! We can set a preconditioner for the initialization. The only available option is LTS. Default NONE.
```

USING EXPLICIT SMOOTHERS

The crucial part of the FAS is smoother, which is essentially the solver. The smoother is set via this flag:

```
mg smoother = STRING! this flag specifies the smoother for FAS. The available options for explicit FAS are: Euler, RK3, RK5, RKOpt.
```

Additionally, for any of the options mentioned above, we can choose to have local time stepping. This is done via this flag:

```
mg preconditioner = STRING ! this flag sets the preconditioner for explicit smoother. The only available option is LTS. Default is NONE.
```

Last, but not least, specifically for the steady-state optimized runge-kutta smoother mg smoother = RKOpt, we have additional option of specifying the number of stages:

```
rk order = INTEGER! this flag sets the number of stages for RKOpt. Default is 5.
```

USING SEMI-IMPLICIT SMOOTHERS

If we want to use a higher CFL, we need an implicit smoother. We set it via the same flag that sets an explicit smoother:

```
mg smoother = STRING ! this flag specifies the smoother for FAS. The available options for implicit FAS are: BIRK5, ILU, SGS.

k gauss seidel = INTEGER ! this flag specifies number of inner SGS iterations for the SGS. Default is 1, which recovers LU-SGS scheme.
```

For the semi-implicit smoothers, we need to define how to compute the Jacobian as well. Necessary flags:

```
#define jacobian

type =INTEGER! this flag specifies how the Jacobian is computed. 1 is numerical, 2 is analytical.

print info = LOGICAL! Set .true. or .false. to print the details regarding the Jacobian computation.

preallocate = LOGICAL! Set .true. or .false. to preallocate memory for the Jacobian.

#end


compute jacobian every = INTEGER! This flag specifies the interval (number of iteration) how often do we re-compute the Jacobian.
```