GAMS/PIPS with SP-SIMPLE and SIMPLE LP with Linking Constraints

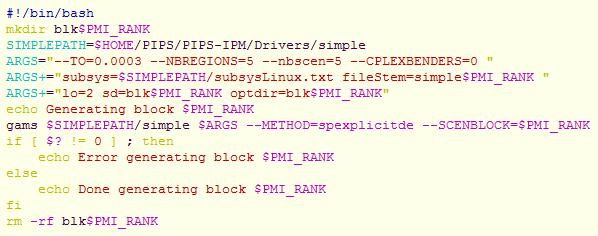
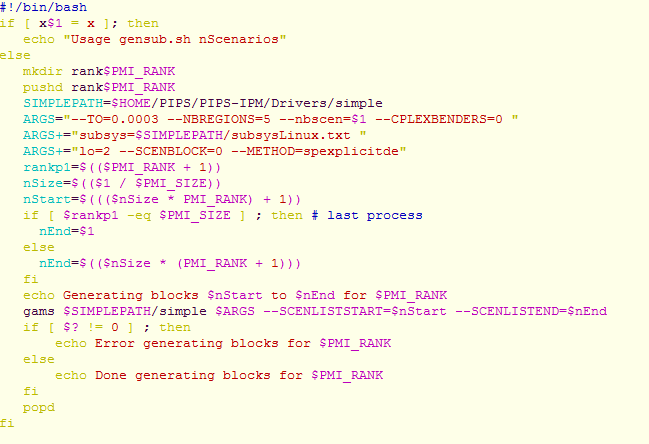
Preparations

1. Install GAMS (current release 24.8). Go to www.gams.com/download and download the installation file that matches your OS. Follow the installation instructions at https://www.gams.com/latest/docs/userguides/userguide/\_u\_g\_\_i\_n\_s\_t\_a\_l\_l.html
2. Install a GAMS license. Copy the following 5 lines into a text file named gamslice.txt in the GAMS system directory (make sure the gamslice.txt has read permissions for the ones that execute gams)  
     
   BEAM-ME\_Project\_License\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_G170330/0001CB-GEN  
   Can\_be\_used\_by\_all\_BEAM-ME\_project\_partners\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
   1433065501ECATBACOCPDEDIGLIPKNLGLDLSM5MBGEOQPTSBSC0SSNXAXP\_\_\_\_\_\_\_  
   1177830609090909090909090909090909090909090909090909090909\_\_\_\_\_\_\_  
   DC12752\_\_\_\_\_\_l\_1,\_m\_0,\_b\_0,\_q\_0,\_p\_0,\_d\_1,\_g\_7\_C\_Eval\_\_\_\_\_\_\_\_\_\_\_\_
3. Adjust the CONVERTD solver configuration to always transport the stage attribute. Edit the text file gmscmpNT.txt (Windows) or gmscmpun.txt (Unix). Change the line  
     
   CONVERTD 101011 5 0001020304 1 1 2 LP MIP ...   
     
   to  
     
   CONVERTD 102015 5 0001020304 1 1 2 LP MIP ...   
     
   Only two digits in the number following the name CONVERTD need to change.

Running SP-SIMPLE (Stochastic Problem without linking constraints)

1. The goal is to generate input files to run the GAMS/PIPS solver. For the experiments, it might be best to add the GAMS system directory to the PATH environment variable. All experiments are done from the command line/shell.
2. Clone the branch "gams-link" from the git repository[[1]](#footnote-1). For sake of simplicity we assume the clone is located in $HOME/PIPS. The relevant folders for the GAMS/PIPS link are in $HOME/PIPS/PIPS-IPM/Drivers. You will find three subdirectories: simple, gmspips, and gams. SIMPLE has all the GAMS and data files required to run GAMS to produce input files for GAMS/PIPS. gmspips contains the custom source for the GAMS/PIPS link while gams contains some C header and source files that are required to build the GAMS/PIPS link but are also shipped with every GAMS distribution.
3. The first experiment is to run a small simple model and solve with Cplex. For this and all other experiments change directory to $HOME/PIPS/PIPS-IPM/Drivers/simple (cd $ HOME/PIPS/PIPS-IPM/Drivers/simple). Run  
     
   gams simple --TO=0.0003 --nbregions=5 --method=spexplicitde --nbscen=5 --CPLEXBENDERS=1  
     
   This will terminate with   
     
   Optimal solution found.

Objective : 11.361938  
  
and some more GAMS lines.

1. Second experiment is to create a single GDX file that contains the entire simple model. Run  
     
   gams simple --TO=0.0003 --nbregions=5 --method=spexplicitde --nbscen=5 --CPLEXBENDERS=0 --SCENBLOCK=-2  
     
   This produces a file called blockall.gdx. Check for the existence and contents of blockall.gdx by running  
     
   gdxdump blockall.gdx  
     
   This file should also list the different block numbers. Run  
     
   gdxdump blockall.gdx symb=x | grep SCALE  
     
   and you should see a couple of lines with SCALE 2 to SCALE 6 in the output. If you do not see any lines, you need to verify that the configuration of the CONVERTD solver (see Preparations 3) has been done properly.
2. Third experiment to run simple with serial block model generation. Run  
     
   gams simple --TO=0.0003 --nbregions=5 --method=spexplicitde --nbscen=5 --CPLEXBENDERS=0 --SCENBLOCK=0  
   gams simple --TO=0.0003 --nbregions=5 --method=spexplicitde --nbscen=5 --CPLEXBENDERS=0 --SCENBLOCK=1  
   gams simple --TO=0.0003 --nbregions=5 --method=spexplicitde --nbscen=5 --CPLEXBENDERS=0 --SCENBLOCK=2  
   gams simple --TO=0.0003 --nbregions=5 --method=spexplicitde --nbscen=5 --CPLEXBENDERS=0 --SCENBLOCK=3  
   gams simple --TO=0.0003 --nbregions=5 --method=spexplicitde --nbscen=5 --CPLEXBENDERS=0 --SCENBLOCK=4  
   gams simple --TO=0.0003 --nbregions=5 --method=spexplicitde --nbscen=5 --CPLEXBENDERS=0 --SCENBLOCK=5  
     
   This produces files called block0.gdx to block5.gdx. Check for the existence of these files and that the SCALE field has the appropriate values.
3. On a HPC (Linux) machine with mpirun prepare a submission script for the parallel block model generation (generateSimpleBlock.sh):  
     
   Execute the generation script (from anywhere):  
     
   mpirun -n 6 $HOME/PIPS/PIPS-IPM/Drivers/simple/generateSimpleBlock.sh  
     
   After the run make sure that the files block0.gdx to block5.gdx exist. The run can be done from any directory and it will deposit the GDX block files in this directory. All temporary and permanent files will be written in this directory. The generateSimpleBlock.sh script knows about the location of simple.gms and other files via the SIMPLEDIR shell variable.
4. Generating bigger instances: The GAMS simple model is highly parameterized. The decomposition in the examples above is done by price scenarios for generation cost (not by regions or time as discussed previously). This decomposition has the advantage to have equal block sizes plus the ability to produce any number of blocks. You can increase the number of blocks by increasing the number after --nbscen. This number corresponds to the number of blocks. You can also increase the size of the individual bock by
   1. increasing the time horizon. For this, one needs to increase the number after the --TO. The maximum number is 1 corresponding to a time horizon of 8760 hours. 0.5 means e.g. 4380 hours.
   2. increasing the number of regions. For this, one needs to increase the number after --nbregions. We have experimented with numbers up to 300. If we want to experiment with a data instance with a *new* number of regions, we need to create the base data for such an instance. This is usually done within the simple script. When we do parallel block model generation and the corresponding base data GDX file is not present the "first" simple run will try to create this and we get a race condition. Hence, we need to generate the base data GDX file before we start the parallel block model generation. This is done by the following run:  
        
      gams simple\_data\_gen.gms --NBREGIONS=*n*in the simple directory.
5. When generating bigger instances we should have in mind how to organize the solution process with GAMS/PIPS. An instance with *n* scenarios (meaning *n+1* blocks) can be solved with at most *n* MPI processes. We can also solve with fewer MPI processes, e.g. *m < n*. The distribution of blocks to MPI processes follows a strict logic in PIPS. MPI process 0 to *m-2* gets *floor(n/m)* scenario blocks plus block 0. The last MPI process m-1 gets the remaining *(n-floor(n/m)\*(m-1))* scenario blocks plus block 0. Here is an example with *n=100* and *m=30*:  
     
    MPI rank 0: blk: 0,1,2,3  
    MPI rank 1: blk: 0,4,5,6  
    MPI rank 2: blk: 0,7,8,9  
    ...  
    MPI rank 28: blk: 0,85,86,87  
    MPI rank 29: blk: 0,88,89,90,...,100  
     
   Knowing the distribution, we can already generate the blocks in a parallel fashion in appropriate subdirectories using the following script (gensub.sh). This script creates permanent directories rank*i* that contain the GDX files of the blocks required by PIPS MPI process *i*:   
     
   mpirun -n 30 $HOME/PIPS/PIPS-IPM/Drivers/simple/gensub.sh 100  
     
   Again, the script can be run from anywhere and creates the rank*i* directories in the current directory. With the example from above (*n=100* and *m=30*) we get:  
     
   [beamme@anton tmp]$ ls  
   rank0 rank11 rank14 rank17 rank2 rank22 rank25 rank28 rank4 rank7  
   rank1 rank12 rank15 rank18 rank20 rank23 rank26 rank29 rank5 rank8  
   rank10 rank13 rank16 rank19 rank21 rank24 rank27 rank3 rank6 rank9  
     
   [beamme@anton tmp]$ ls rank0 rank1 rank2 rank28 rank29  
   rank0:  
   block0.gdx block2.gdx convertd.opt simple.lst  
   block1.gdx block3.gdx simple.log  
   rank1:  
   block0.gdx block5.gdx convertd.opt simple.lst  
   block4.gdx block6.gdx simple.log  
     
   rank2:  
   block0.gdx block8.gdx convertd.opt simple.lst  
   block7.gdx block9.gdx simple.log  
     
   rank28:  
   block0.gdx block86.gdx convertd.opt simple.lst  
   block85.gdx block87.gdx simple.log  
     
   rank29:  
   block0.gdx block89.gdx block92.gdx block95.gdx block98.gdx simple.log  
   block100.gdx block90.gdx block93.gdx block96.gdx block99.gdx simple.lst  
   block88.gdx block91.gdx block94.gdx block97.gdx convertd.opt  
     
   The gensub.sh script:  
     
   

Running SIMPLE4PIPS (“Standard” LP with linking constraints)

1. The goal is to generate input files to run the overhauled GAMS/PIPS solver that supports linking constraints. For the experiments, it might be best to add the GAMS system directory to the PATH environment variable. All experiments are done from the command line/shell.
2. Clone the branch "gams-link" from the git repository[[2]](#footnote-2). For sake of simplicity we assume the clone is located in $HOME/PIPS. The relevant folders for the GAMS/PIPS link are in $HOME/PIPS/PIPS-IPM/Drivers. You will find three subdirectories: simple, gmspips, and gams. SIMPLE has all the GAMS and data files required to run GAMS to produce input files for GAMS/PIPS. gmspips contains the custom source for the GAMS/PIPS link while gams contains some C header and source files that are required to build the GAMS/PIPS link but are also shipped with every GAMS distribution.
3. The first experiment is to run a small simple model and solve with Cplex. For this and all other experiments change directory to $HOME/PIPS/PIPS-IPM/Drivers/simple (cd $ HOME/PIPS/PIPS-IPM/Drivers/simple). Run  
     
   gams simple4pips --NBREGIONS=5 --TO=0.0191 lp=cplex  
     
   This will terminate with   
     
   Optimal solution found.

Objective : 891.807988  
and some more GAMS lines.

1. Second experiment is to create a single GDX file that contains the entire simple model with annotation:

gams simple4pips --NBREGIONS=5 --TO=0.0191 --METHOD=PIPS --TBSIZE=24

The instance created via parameters --NBREGIONS=5 --TO=0.0191 contains 5 regions and 168 hours (=1 week). Parameter --TBSIZE=24 means that 24 time steps form a time block, i.e. we obtain 7 time blocks plus the block of linking variables. If TBSIZE is no divisor of the number of time steps the last time block will be smaller than TBSIZE time steps.  
The call above produces a file called allblocksPips.gdx. Check for the existence and contents of allblocksPips.gdx by running  
  
gdxdump allblocksPips.gdx noData  
  
This file should also list the different block numbers. Run  
  
gdxdump allblocksPips.gdx symb=x | grep SCALE  
  
and you should see a couple of lines with SCALE 2 to SCALE 8 in the output that indicate the block membership of the variables. If you do not see any lines, you need to verify that the configuration of the CONVERTD solver (see Preparations 3) has been done properly.

1. The next experiment is to split allblocksPips.gdx into several block files. There is a tool gmspipschk that checks for consistency of the annotation and can also be used to split such a file. How to compile that tool is explained below in section [Running GAMS/PIPS](#RunningGAMSPIPS).

$HOME/PIPS/build/gmspipschk -t -X 8 allblocksPips.gdx  
  
This produces files called allblocksPips0.gdx to allblocksPips7.gdx. Check for the existence of these files and that the SCALE field has the appropriate values. For example, with

gdxdump allblocksPips3.gdx FilterDef=N symb=x | grep SCALE

you should see only variables with SCALE=4 (those belong to block 3) and SCALE=1 (linking variables).

Note that gmspipschk is parametrizable and can be used for different purposes. For a brief summary on how to use gmspischk run

$HOME/PIPS/build/gmspipschk -h

The following should be written to output:

Usage: [-hsSptTxXdD] [-g GAMSSysDir] [-b actBlock] [-o n] numBlocks dirStem

-h print usage

-s strict mode

-S very strict mode

-p GNUPlot output to stdout, cmd= 'plot "file" using 2:1 with dots'

-t split GDX file into multiple GDX files

-T split GDX file into multiple GDX files without uels and strings

-w output of block structure counts to stdout

-W output of block structure to stdout

-x dirStem is GDX file stem

-X dirStem is GDX file

-d dirStem is scratch directory stem (default)

-D dirStem is scratch directory

-g specify GAMS system directory

-b specify single block

-o specify stage offset (default 1)

numblocks total number of blocks

dirStem scratch directory name stem

1. Generating bigger instances: The GAMS model simple4pips is highly parameterized. The decomposition in the examples above is done by time blocks. However, simple4pips also supports regional decomposition as well as a combination of regional and time decomposition. As regional decomposition does not appear to be very promising, the default is to decompose (annotate) by time blocks only. By default, a time block contains 24 time steps. The size can be varied via parameter --TBSIZE. The number of time blocks can be increased by
   1. increasing the time horizon. For this, one needs to increase the number after the --TO. The maximum number is 1 corresponding to a time horizon of 8760 hours. 0.5 means e.g. 4380 hours.
   2. decreasing the resolution. For this, one needs to set the number after optional parameter --RESOLUTION to a smaller value. Default is 1 which corresponds to a time discretization of 1 hour. Setting --RESOLUTION=0.5 results in 30 minutes steps etc.
   3. decreasing the time block size. For this, one needs to set the number after --TBSIZE to a smaller value.

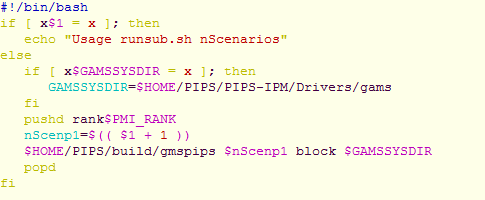
To increase the entire model one could also increase the number of regions. For this, one needs to increase the number after --NBREGIONS. If we want to experiment with a data instance with a *new* number of regions, we need to create the base data for such an instance.

**Note that the parallel generation of gdx block files** [as illustrated for the Stochastic Problem](#parallelModelGen) **has not yet been implemented in simple4pips. In case the distribution of files to the different** [ranki directories](#rankiFolders) **as illustrated for the Stochastic problem should be mimicked, currently on would need to do that manually, or create a little script for that task. Then one could also use a script similar to** [runsub.sh](#runsub) **start gmspips in such a setup.**

Running GAMS/PIPS

1. First we need to compile gmspips. For this (and the program gmspipschk) lines for compilation instructions have been added to the CMakeList.txt in the $HOME/PIPS/PIPS-IPM folder. Now change the directory to the $HOME/PIPS/build folder and run " cmake .. -DBUILD\_ALL=OFF -DBUILD\_PIPS\_IPM=ON && make". This should produce an executable gmspips in the build folder.
2. Now we are ready to run gmspips in the folder where we have placed all GDX block files:  
     
   mpirun -n 5 $HOME/PIPS/build/gmspips 6 /path/to/gdx/block $GAMSSYSDIR  
     
   Please note that the first argument of gmspips (6) is the total number of blocks. If we generate for example the stochastic problem version with 5 scenarios (--NBSCEN=5) we obtain an instance with 6 blocks in total. However, the maximum number of MPI tasks is the total number of blocks **minus 1**. For 5 scenarios this would again correspond to 5 as in mpirun -n 5 […].

The environment variable $GAMSSYSDIR points to the GAMS system directory location. You can start gmspips in any directory. If the GDX files are not located in your current directory, you need to specify the location to the GDX file as part of the file name stem.

In the more complicated setup where the block GDX files are placed in the rank*i* directories gmspips needs to be started by a script (runsub.sh) because the location of the GDX files depends on the PMI\_RANK variable:  
  
  
This script runsub.sh needs to be started in the directory that contains all rank*i* directories:   
  
mpirun -n 30 $HOME/PIPS/PIPS-IPM/Drivers/simple/runsub.sh 100

1. gitlab.version.fz-juelich.de/breuer1/PIPS\_beamme [↑](#footnote-ref-1)
2. gitlab.version.fz-juelich.de/breuer1/PIPS\_beamme [↑](#footnote-ref-2)