GIDGET USER GUIDE

JOHN FORBES

1. Introduction

This document and the code itself is a work in progress.

Please email jforbes at ucolick.org with questions, concerns, issues, or comments.

Now that you've downloaded the source code, it should be easy to get going running your own 1D disks.

The source code is located in the src directory. You are free to modify and re-distribute the code under the GNU General Public License (GPL), [1] For more details see the LICENSE file in this directory. We ask that you cite Forbes, Krumholz, and Burkert (2012) and/or Forbes et al (2013), depending on which version of the code you have been using, if you publish work based on this code.

The code relies on the GNU Scientific Library (GSL). Header files are included in gidget/inc/gsl, and a static version of the library is provided in the src directory. This setup allows the binary to be run on machines which do not have their own version of GSL installed, e.g. on a Mac Xgrid.

Some documentation is provided in the source code, particularly the header files in gidget/src.

2. Installation

You can obtain the source (presumably you've already done so if you're reading this) from `http://www.ucolick.org/~jforbes/gidget.html` or (RECOMMENDED) the linked bitbucket.org page. To compile and run the code you'll probably need

(1) gcc
(2) the standard C++ library

The following things would be very helpful, but are not strictly necessary to run GIDGET

(1) git
(2) python
(3) numpy
(4) IDL
(5) The astronomy user's library for IDL (`http://idlastro.gsfc.nasa.gov/`)
(6) The Coyote Graphics IDL library (`http://www.idlcoyote.com/documents/programs.php`)

[1] available in full at `http://www.gnu.org/copyleft/gpl.html`

(7) Dependencies for latexify by Robert da Silva (`http://slugidl.pbworks.com/w/page/37657460/latexify%20tutorial`)

To install, just change directories to gidget/src, and run make. This should produce an executable gidget/bin/gidget which you could run directly from the command line, though I recommend using the python interface (see next section). You should also make sure that the IDL libraries and the directory gidget/idl are in your IDL path.

3. Running

GIDGET has a fair number of parameters. These range from physical parameters (the circular velocity of the simulated galaxy) to choices about the computational domain (how many radial cells should the code use?), to experimental options where I want to quickly try out something new without adding a whole new parameter (solve the advection part of the energy equation using the upstream derivative rather than a minmod slope limiter). The physical and computational parameters are summarized in table 1 of the latest paper, reproduced in a potentially more helpful form here.

To run the code, I would recommend editing the file gidget/exper.py. This is a python script which is especially useful for running sequences of experiments where some physical parameter is systematically varied. You can also easily set up experiments where multiple parameters are varied together (for example halo mass, mass loading factor, circular velocity, radius,...), or even experiments where many parameters are varied independently. There are many examples in the script. I'll go through a few of them here:

```
# Guess for a reasonable model.
l045 = GetScaleLengths(1,Mh0=1.0e12,scatter=1.0e-10)[0]
print "l045 = ",l045," kpc"
rg01=experiment("rg01")
rg01.irregularVary("R",40)
rg01.irregularVary('accScaleLength',l045*.7)
rg01.irregularVary('mu',.5)
rg01.irregularVary('vphiR',220.0)
rg01.irregularVary('NPassive',20)
rg01.irregularVary('invMassRatio',1.0)
rg01.irregularVary('dbg',2)
rg01.irregularVary('xmin',.002)
rg01.irregularVary('alphaMRI',.01)
rg01.irregularVary('fcool',0.6)
rg01.irregularVary('innerPowerLaw',0.5)
rg01.irregularVary('b',3.0)
rg01.irregularVary('nx',200)
rg01.irregularVary('kappaMetals',1.0)
rg01.irregularVary('xiREC',0.0)
rg01.irregularVary('alphaAccretionProfile',1./3.)
rg01.irregularVary('deltaOmega',.1)
```

TABLE 1. Code parameters.

Parameter	Fiducial Value	Name in exper.py	Description
			Gas Migration
η	1.5	eta	(3/2) kinetic energy dissipation rate per scale-height crossing time
Q_{GI}	2	fixedQ	Marginally stable value of Q
$T_{\rm gas}$	7000 K	gasTemp	Gas temperature; sets the minimum gas velocity dispersion
α_{MRI}	0.01	alphaMRI	Value of $T_{r\phi}/\rho\sigma_{th}^2$ without gravitational instability
			Rotation Curve
$v_{\rm circ}$	220 km s^{-1}	vphiR	Circular velocity in flat part of rotation curve
r_b	3 kpc	b	Radius where rotation curve transitions from powerlaw to flat
β_0	0.5	innerPowerLaw	Powerlaw slope of $v_\phi(r)$ at small radii
n	2	softening	Sharpness of the transition in the rotation curve
			Star Formation
$\epsilon_{\rm ff}$	0.01	epsff	Star formation efficiency per freefall time in the Toomre regime
$f_{\rm H_2,min}$	0.03	fH2Min	Minimum $f_{\rm H_2}$.
t_{SC}	2 Gyr	tDepH2SC	Depletion time of H_2 in the single cloud regime
f_R	0.54	RfREC	Mass fraction of a zero-age stellar population not recycled to the ISM
μ	0.5	mu	Galactic winds' mass loading factor
			Metallicity
y	.054	yREC	Mass of metals yielded per mass locked in stellar remnants
ξ	0	xiREC	Metallicity enhancement of galactic winds
Z_{IGM}	$0.1Z_\odot = 0.002$	ZIGM	Metallicity of initial and infalling baryons
k_Z	1	kappaMetals	Amplitude of metallicity diffusion relative to Yang and Krumholz (2012)
			Stellar Migration
$Q_{\rm lim}$	2.5	Qlim	Value of Q_* below which spiral instabilities will heat the stars
$T_{\rm mig}$	4	tauHeat	Number of local orbital times over which stars are heated by spiral instabilities
			Accretion
$M_{h,0}$	$10^{12}M_\odot$	Mh0	Halo mass at $z = 0$
$\Delta\omega$	0.5	deltaOmega	Interval of $\omega \sim z$ over which accretion rate is constant [2]
$r_{\rm acc}(z=0)$	6.9 kpc	accScaleLength	Scale length of new infalling gas
β_z	0.38	accAlphaZ	Scaling of efficiency with $(1+z)$
β_{M_h}	-0.25	accAlphaMh	Scaling of efficiency with halo mass
ϵ_0	0.31	accNorm	Efficiency at $M_h = 10^{12}M_\odot$, $z = 0$
ϵ_{max}	1	accCeiling	Maximum value of efficiency
			Initial Conditions
α_r	1/3	alphaAccretionProfile	Scaling of accretion scale length with halo mass
$f_{g,0}$	0.5	fg0	Initial gas fraction
f_{cool}	1	fcool	Fraction of $f_b M_h(z = z_{relax})$ contained in the initial disk
z_{relax}	2.5	zrelax[3]	z at which the simulation is initialized
ϕ_0	1	phi0	Initial ratio of stellar to gaseous velocity dispersion
			Computational Domain
x_0	.004	*xmin*	Inner edge of domain as a fraction of R
R	40 kpc	R	Outer edge of domain
n_x	200	nx	Number of radial cells
tol	10^{-4}	TOL	Fastest change allowed in state variables, per orbital time at $r = R$
			Other
-	1	analyticQ	Use Romeo and Wiegert (2011), not Rafikov Q (Setting this to zero may not work as expected)
-	1	cosmologyOn	Evolve the redshift. (Setting this to zero may not work as expected)
-	5000	tmax	Maximum number of outer orbits to evolve (usually sim terminates at $z = 0$)
-	10^9	stepmax	Maximum number of timesteps to take
-	1.5	thickness	Thickness correction to Q_{gas}
-	1	migratePassive	Solve stellar migr. eqns. for passive populations
-	5.0 km/s	minSigSt	A floor for stellar velocity dispersions.
-	3	NChanges	Number of times to draw a new number from lognormal distr. to generate accr. history for neg. values of whichAccretionHistory
-	200	Noutputs	Number of snapshots to output (equally spaced in time), plus 1 for the initial conditions and 1 for final conditions
-	0	dbg	Bitwise activate various 'experimental' switches in the code (see Main.cpp for a current list)
-	0	whichAccretionHistory	0 is the smooth accr. history, large positive values are random seeds for Neistein and Dekel '08 accretion histories, large negative values are random seeds to generate a purely lognormal distribution of accr. histories preserving the avg. growth in the smooth accr. history.
-	0	whichAccretionProfile	The radial profile of $\dot{\Sigma}_{cos}$: 0- exponential scale length, 1- flat, 2- gaussian around some radius. The scale of all of these profiles is given by accScaleLength
-	0.1	widthAccretionProfile	For gaussian accr. profile, the width of the distribution as a fraction of its center.

Here we are defining an experiment named rg01. This is a python object, but it will also be the basename for the directory and files that will eventually be produced by the code. In the first line, we define l045, a scale length derived using the assumption of a median value in the spin parameter. We then create the experiment object - the constructor takes a string argument which will be the basename of the output files. The next lines set various parameters, which should be recognizable in Table 1. The irregularVary function can take a single value as its second argument, or a Python list. The former simply sets the given parameter to the given value (overriding the default), while the second will run a sequence of gidget models, each with the given parameter set to the next element in the list. If multiple parameters are given lists in this manner, each of those variables will be varied independently. To avoid that and have them vary together, a third argument to irregularVary must be provided (see the next example).

```
# Vary the scale length of the accretion.
rg02=NewSetOfExperiments(rg01,"rg02",N=2)
rg02[0].vary('accScaleLength',l045*.10,l045*.67,5,0,3)
rg02[0].vary('R',l045*.10*7,l045*.67*7,5,0,3)
rg02[1].vary('accScaleLength',l045*.73,l045*2.1,10,0,3)
rg02[1].vary('R',l045*.73*7,l045*2.1*7,10,0,3)
```

Here, instead of manually creating a new experiment object with the experiment constructor, we call NewSetOfExperiments, which takes a previously-defined experiment, copies it N=2 times, and sets a new base name, "rg02". This time the rg02 object is not an experiment object, but a list of two experiment objects. Having defined this list, we then vary accScaleLength and R together. This time we use vary instead of irregularVary, so that we can set up an equally-spaced grid. The second line of this example varies accScaleLength from one tenth of l045 to .67 of l045 in 5 steps (that's the meaning of the first 4 arguments). The 5th argument (0) tells vary to space the values linearly (use 1 for logarithmic), and the final value (3), tells vary that this variable is to vary together with any other variable which also has a '3' as the 6th argument of vary or the 3rd argument of irregularVary. Not coincidentally, the next line, which tells the first experiment of rg02 to also vary R, has a '3' as well, meaning that accScaleLength and R are to vary in lock-step. This requires that both variables have the same number of steps (5 in this case). The next two lines do the same steps but for the second experiment in rg02.

OK, suppose you have some experiment(s) defined. How do you run them? This is the easy part. Change to the top-level gidget directory and run:

```
$ python exper.py --nproc 4 rg01 rg02a rg02b
```

will run your three experiments (each of which may in principle contain many GIDGET simulations) on 4 processors on your local machine. The output files will be stored in gidget/analysis/rg01, gidget/analysis/rg02a and gidget/analysis/rg02b. The arguments to exper.py are automatically expanded, so that the above line is equivalent to

```
$ python exper.py --nproc 4 rg01 rg02
```

and similarly the following line

```
$ python exper.py --nproc 4 rg0
```

will run every experiment whose name contains 'rg0'. exper.py will not overwrite experiments you have already run, so to re-run an experiment you should manually delete its output directory first.

4. Analysis

You have the option to use IDL or python for analysis. The python interface is much more user-friendly. Here are a few examples using plots.py, to be run in the gidget/py directory. (Of course you have to have defined an experiment in exper.py and run it as above before you analyze it):

```
$ python plots.py rg01 --balance
```

produces a 'balance' movie like the frames in figure 3 of Forbes et al (2013). To create a movie of some radial quantity for all the models in an experiment, you can use for example

```
$ python plots.py rg02 --radial --vsr col Z colst --colorby accScaleLength
```

This will create movies of col (gas column density), Z (metallicity), and colst (stellar column density), where each frame will be a plot of this quantity vs. radius at a given time in the simulation, and the individual models will be colored by its accScaleLength (accretion scale length).

Other useful options for plots.py include –scaled (like –radial only plot vs r/rAcc, the radius in units of the current accretion scale length), –mass (plots of integrated quantities vs. stellar mass, stiched into movies), –time (plot integrated quantities vs. time). The .mov and .png files will appear in the folder gidget/py/¡experimentName¿. The radial- and time- dependent quantities are defined in gidget/py/readoutput.py, where they are constructed from the raw data produced by the C++ code.

4.1. **IDL.** Here's how you do analysis with IDL (not recommended!). This is done mostly through a few 'scripts' which generate obscene numbers of plots. I tend to think of these as scripts since they are changed frequently when making minor alterations to plots, etc. The source code for the scripts and all of the underlying code can be found in the gidget/idl directory. The script I use most is called variability3, which takes as an argument a list of experiments. For every parameter to be graphed, e.g. radius vs. column density, velocity dispersion, metallicity,.... and time vs. r25, stellar mass, SFR,..., the code creates plots with all the models from each experiment colored the same way. For instance,

```
IDL> variability3,['rg01','rg02a','rg02b']
```

creates crp_rg01_rg02a_rg02b_15_unsortedLogR_col_timeSeries.eps, which is shown in figure 1. If you run experiments rg01, rg36, and rg69b, all defined in exper.py, you can in fact recreate figures 1-15 of the current paper, mostly with variability3. To do so, you can run

```
$ python exper.py --nproc 4 rg01 rg36 rg69b
$ idl
IDL> allplotsf13a,[0]
IDL> exit
```

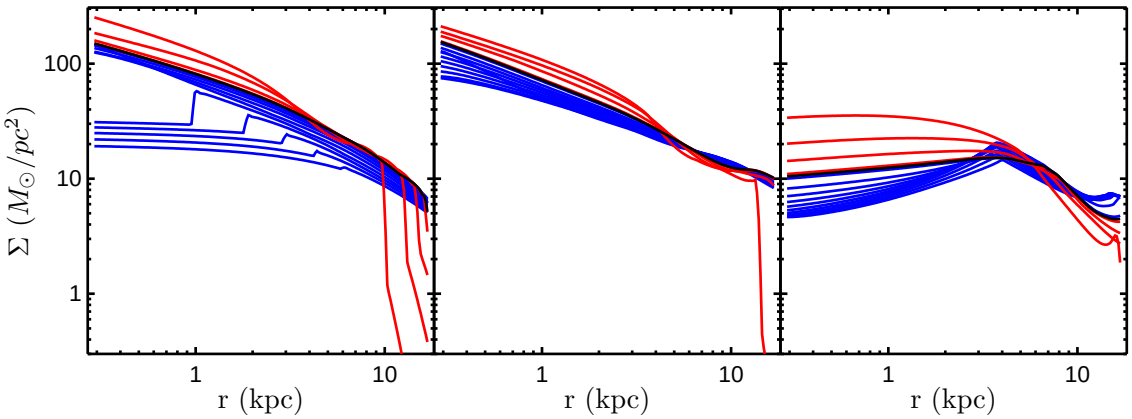

FIGURE 1. Time evolution of the column density profile for experiments rg01 (black), rg02a (red), and rg02b (blue).

```
$ idl
IDL> allplotsf13a,[1]
IDL> exit
$ idl
IDL> allplotsf13a,[2]
IDL> exit
$ idl
IDL> allplotsf13a,[4]
IDL> exit
$ idl
IDL> allplotsf13a,[5]
IDL> exit
$
```

This creates many figures, among them

```
crp_rg01_rg69b_401_vstimeDistGroupFill_Mh_timeSeries.eps
crp_rg01_rg36d_rg36e_3_unsortedSP_col_timeSeries.eps
crp_rg01_rg36d_rg36e_balance_col_all.eps
crp_rg01_balance_sig_only.eps
crp_rg01_rg36e_2_unsortedSPGroup_colsfr_timeSeries.eps
crp_rg01_rg36d_rg36e_3_unsortedSP_colPerCrit_timeSeries.eps
crp_rg01_rg36d_2_unsortedSP_mdotDisk_timeSeries.eps
crp_rg01_rg69b_401_unsortedDistFill_col_timeSeries.eps
crp_rg01_rg69b_401_unsortedDistFill_equilibrium_timeSeries.eps
crp_rg01_rg69b_401_vstimeDistGroupFill_r25d_timeSeries.eps
crp_rg01_rg69b_401_vstimeDistGroupFill_maxSig_timeSeries.eps
crp_rg01_rg69b_401_unsortedDistFill_BB4_timeSeries.eps
```

`crp_rg01_rg69b_401_unsortedDistFill_sig_timeSeries.eps`
`crp_rg01_rg69b_401_vstimeDistFill_BTexc_timeSeries.eps`
`crp_rg01_rg69b_401_unsortedDistFill_MJeans_timeSeries.eps`
which are the figures in the paper.