GalPaK3D: Galaxy Parameters and Kinematics

Fernanda Muñoz - Python Meeting

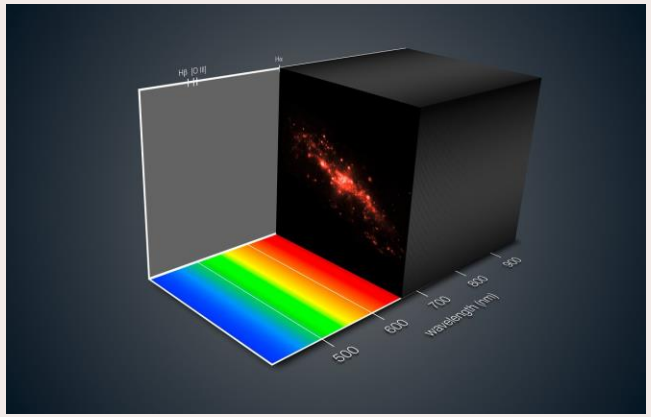
Agenda

1. What is GalPaK3D?
2. Parameter meaning
3. How to use it
4. Examples
5. Q&A

What is GalPaK 3D?

- Tool to extract the intrinsic (i.e. deconvolved) Galaxy Parameters and Kinematics from any **3-Dimensional data.**
- The algorithm directly compares data-cubes with a disk parametric model which has 9 or 10 free parameters, using a Markov Chain Monte Carlo (MCMC) approach.
- Flexible enough to handle any instrument.

The algorithm can be summarized as:



The input :)

Pick new random galaxy parameters using uniform priors (boundaries can be set)

Create clean cube from parameters and convolve it with instrument's PSF+LSF

Measure closeness of resulting convolved cube to input cube

Accept/Reject galaxy parameters using Metropolis-Hasting algorithm

Goto 1. until max iterations are reached

Fill output attributes with data (chain, cubes, etc.)

Compute and return best fit galaxy parameters from chain!!

```
pip install galpak
```

Parameter meaning
Galaxy Parameters

Warning: the velocity_dispersion parameter is NOT the total dispersion. This parameter is akin to a turbulent term. It is added in quadrature to the dispersions due to the disk model and to the thickness.

Component	Description
gp.x	X coordinates of the center of the galaxy. (in pixels)
gp.y	Y coordinates of the center of the galaxy. (in pixels)
gp.z	Z coordinates of the center of the peak. (in pixels)
gp.flux	Sum of the values of all pixels. (in the unit of the cube)
gp.radius	Radius of the galaxy. (in pixels)
gp.inclination gp.pitch	Inclination of the galaxy (in degrees), aka. pitch along observation axis.
gp.pa gp.roll	Position Angle, clockwise from Y (in degrees), aka. roll along observation axis.
gp.rv gp.turnover_radius	Inverse factor rv in expression *arctan(r/rv)*.
gp.maximum_velocity	Maximum Velocity. (in km/s)
gp.velocity_dispersion gp.sigma0	Disk Dispersion spatially constant (in km/s) added in addition to kinematics dispersion.

Parameter meaning
Galaxy Parameters

Access and fixed parameters

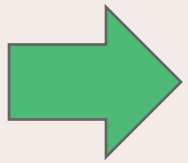
```
pip install galpak
```

Component	Description
gp.x	X coordinates of the center of the galaxy. (in pixels)
gp.y	Y coordinates of the center of the galaxy. (in pixels)
gp.z	Z coordinates of the center of the peak. (in pixels)
gp.flux	Sum of the values of all pixels. (in the unit of the cube)
gp.radius	Radius of the galaxy. (in pixels)
gp.inclination gp.pitch	Inclination of the galaxy (in degrees), aka. pitch along observation axis.
gp.pa gp.roll	Position Angle, clockwise from Y (in degrees), aka. roll along observation axis.
gp.rv gp.turnover_radius	Inverse factor rv in expression *arctan(r/rv)*.
gp.maximum_velocity	Maximum Velocity. (in km/s)
gp.velocity_dispersion gp.sigma0	Disk Dispersion spatially constant (in km/s) added in addition to kinematics dispersion.

In [1]:
```python
from galpak import GalaxyParameters

gp = GalaxyParameters(x=1.618)

# Classic numpy.ndarray numeric access
assert(gp[0] == 1.618)
# Autocompleted and documented property access
assert(gp.x == 1.618)
# Convenience dictionary access,
assert(gp['x'] == 1.618)
# ... useful when you're iterating :
for name in GalaxyParameters.names:
    print(name)
    print(gp[name])
```



```
x
1.618
y
nan
z
nan
flux
nan
radius
nan
inclination
nan
pa
nan
turnover_radius
nan
maximum_velocity
nan
velocity_dispersion
nan
```

Parameter meaning
Getting the Wavelength

The z attribute in a GalaxyParameter is in pixels, you may want the value in the physical unit specified in your Cube's header.

To that effect, you may use the *wavelength_of* method of the HyperspectralCube:

```
In [2]: # This is the mock cube
        path = 'galpak/data/input/GalPaK_cube_1101_from_paper.fits'

In [3]: from galpak import GalPaK3D

        gk = GalPaK3D(path)

        wavelength = gk.cube.wavelength_of(gk.galaxy.z)
```

```
INFO:GalPaK:Reading cube from galpak/data/input/GalPaK_cube_1101_from_paper.fits
WARNING:GalPaK:Using the MUSE instrument per default. You should specify your own instrument.
INFO:root:Patching the header with cards provided
WARNING:GalPaK:No variance provided. Estimating Variance from edge statistics
INFO:GalPaK:Computed stdev from the edges: Var=5e-20
INFO:GalPaK:Variance estimated is 4.7287528811180427e-20
INFO:GalPaK:Setting up with the following setup :
psf = Gaussian PSF :
  fwhm         = 1.0 "
  pa           = 0.0 °
  ba           = 1.0
lsf = Gaussian LSF : fwhm = 2.675  Angstrom

cube_xy_step   = 0.2 "
cube_z_step    = 1.25 Angstrom

cube z_step_kms = 57.12979890310786 km/s  at 6564.0 Angstrom
```

Input Cubes supported

- Any fits cube can be used p
  the z-axis represents wavel
  frequencies.
- Any units are normally acce
  algorithm works in velocity
- If the header is incomplete
  CDELT3, CRVAL3, CUNIT3),
  will try to use the default va
  assigned to the instrument.
- If the header is complete, th
  default pixel sizes will be o
  by the the information from the cube
  header.
- MPDAF Obj.Cube object is ok as input.

Instuments supported

	ult) or less or use
	ult lsf_fwhm is 2.67
MANGA	(soon)

2. **Parameter meaning** ○○○●

Level 0

Running the program, default parameters and save files

MAKE SURE YOUR CUBE :

1. Is written as a fits file

2. Has a third dimension representing wavelengths or frequencies.

3. Has a minimum info in the header.
   - The header should have units but the algorithm works in velocity space (dlamba/lamba or dfrequency/frequency).
   - If the header is incomplete (CRPIX3, CDELT3, CRVAL3, CUNIT3), the algorithm will try to use the default values assigned to the instrument.
   You can specify these directly.

4. Is adequately cropped in x and y and along z around the galaxy.

5. Is continuum subtracted (!) as the algorithm does not fit the continuum

6. Best to provide the variance in a separate file or via a MPDAF Obj.Cube. If no variance is specified, the cube statistics will be used.

Warning: You should specify the PSF and spectral LSF that are in effect for your cube.

Run GalPaK examples

Assuming your cube can be read properly, you can run it with :

```
In [7]: gk = GalPaK3D(path, seeing=1.0, instrument=galpak.MUSE(lsf_fwhm=2.51))
        gk.run_mcmc(max_iterations=500)
        gk.save('my_galpak_run')
```

or with :

```
In [30]: my_instrument = galpak.MUSE(lsf_fwhm=2.51)
         gk = galpak.run(path, seeing=1.0, instrument=my_instrument, max_iterations=500)
         gk.save('my_galpak_run') # Specify clobber=True to overwrite it.
```

and you can check the instrument properties with :

```
In [8]: # Check the instrument properties with :
        print(gk.instrument)
```

```
psf = Gaussian PSF :
  fwhm         = 1.0 "
  pa           = 0.0 °
  ba           = 1.0
lsf = Gaussian LSF : fwhm = 2.51  Angstrom

cube_xy_step   = 0.2 "
cube_z_step    = 1.25 Angstrom

cube z_step_kms = 57.12979890310786 km/s  at 6564.0 Angstrom
```

Run GalPaK examples

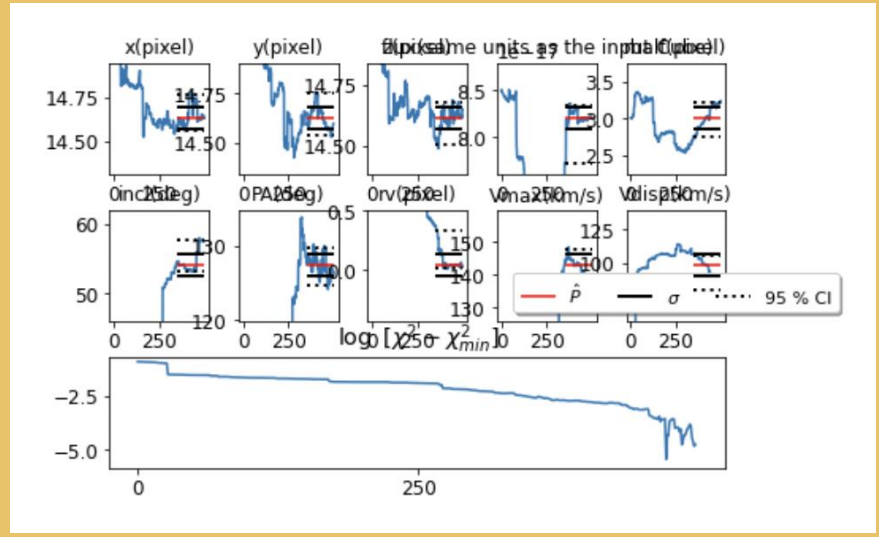
Assuming your cube can be read properly, you can run it with :

```
In [7]: gk = GalPaK3D(path, seeing=1.0, instrument=galpak.MUSE(lsf_fwhm=2.51))
        gk.run_mcmc(max_iterations=500)
        gk.save('my_galpak_run')
```

or with :

```
In [30]: my_instrument = galpak.MUSE(lsf_fwhm=2.51)
         gk = galpak.run(path, seeing=1.0, instrument=my_instrument, max_iterations=500)
         gk.save('my_galpak_run') # Specify clobber=True to overwrite it.
```

Bad results!!!



If you would like to access the results:

```
In [5]: gk.galaxy
```

```
Galaxy Parameters : value (± stdev) [units]     [Confidence Interval]
   x: 14.55 ± 0.09 (pixel)      CI 95%: [14.35,14.74]
   y: 14.54 ± 0.09 (pixel)      CI 95%: [14.37,14.71]
   z: 14.548436 ± 0.069703 (pixel)      CI 95%: [14.422266,14.677856]
   flux: 8.45e-17 ± 2.59e-18 (same units as the input Cube)     CI 95%: [7.93e-17,8.91e-17]
   radius: 4.07 ± 0.17 (pixel)  CI 95%: [3.78,4.41]
   inclination: 71.08 ± 2.58 (deg)      CI 95%: [66.32,75.36]
   pa: 125.98 ± 1.77 (deg)      CI 95%: [122.41,129.29]
   turnover_radius: 1.02 ± 0.31 (pixel)         CI 95%: [0.71,1.76]
   maximum_velocity: 182.68 ± 17.49 (km/s)      CI 95%: [165.42,226.62]
   velocity_dispersion: 81.44 ± 5.57 (km/s)     CI 95%: [71.61,94.39]
```

Warning: seeing=1.0 is equivalent to galpak.MUSE(psf_fwhm=1.0)

The proper way to specify the seeing is specified in customizing-the-psf.

3. How to use it ○○●○○○

Default Model Parameters

Both run and run_mcmc use a *DiskModel* object with exponential flux profile per default, arctangent rotation profile and thick disk dispersion.

Default Model Parameters

You can see the detailed documentation for the available parameters from the model class, but here's a highlight of the most important ones :

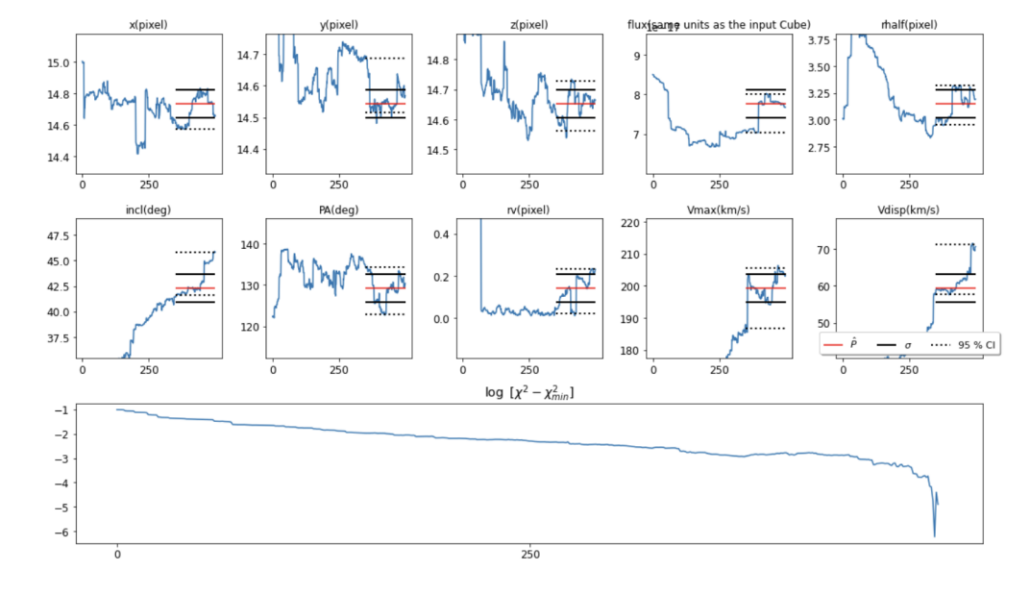
- flux_profile : 'gaussian' or 'exponential' (default) or 'de_vaucouleurs'. The flux profile of the galaxy.
- rotation_curve : The profile of the velocity v(r) can be in:
  - 'isothermal' : a pseudo-isothermal V~Vmax (1-actan(X)/X) where X=r/rV, rV=turnover_radius
  - 'arctan' (default) : V~ Vmax arctan(r/rV), rV=turnover radius
  - 'tanh' : an tanh profile V~Vmax tanh(r/rV)
  - 'exp' : inverted exponential, 1-exp(-r/rV)
  - 'mass' : a constant light-to-mass ratio v_circ(r)=sqrt(G m(<r) / r)
- disk_dispersion : 'thick' or 'thin' (or 'infinitely_thin' TBD).

Default Model Parameters

GalPak has 3 components for the dispersion:

- a component from the rotation curve arising from mixing velocities of a disk with non-zero thickness
- a component from the local disk dispersion specified by run_mcmc.disk_dispersion a spatially constant dispersion, (which is the output parameter gk.galaxy.velocity_dispersion)

Better!!



You should change these options:

```
In [9]: my_instrument = galpak.MUSE(lsf_fwhm=2.51)
        mydisk = galpak.DiskModel(flux_profile='gaussian', rotation_curve='isothermal')
        gk = galpak.run(path, seeing=1.0, instrument=my_instrument, model=mydisk, max_iterations=500)
        #gk.save('my_galpak_run')
```

You can always check your model with:

```
In [10]: print(gk.model)

{'flux_profile': 'gaussian', 'thickness_profile': 'gaussian', 'dispersion_profile': 'thick', 'rotation_curve': 'isothermal',
'line': None, 'redshift': None, 'hz_profile': 'gaussian', 'disk_dispersion': 'thick', 'Parameters': <class 'galpak.galaxy_par
ameters.GalaxyParameters'>, 'GalaxyParameters': <class 'galpak.galaxy_parameters.GalaxyParameters'>, 'logger': <Logger GalPa
K: DiskModel: (INFO)>, 'pixscale': 0.2}
```

Run Output Overview

The run_mcmc method returns a GalaxyParameters object.

Type : print(gk.galaxy)

to see the parameters (gk.error contains the error vector).

Use the methods of GalaxyParameters tofile or tolist to store this.

The run_mcmc also fills the gk instance parameters with several output data :

- gk.deconvolved_cube : best guess as to what the deconvolved cube should be
- gk.convolved_cube : virtually convolved cube, should be close to the inputted measure cube
- gk.residuals_cube : differential between measure cube and convolved cube, inversely scaled by measure error
- gk.psf3d : the 3D PSF*LSF used for the convolution
- gk.chain : the full markov chain, with each step holding its galaxy parameters and reduced χ^2
- gk.acceptance_rate : the final proportion of useful iterations in %
- gk.initial_parameters : the first parameters of the markov chain
- gk.galaxy : a view to the returned GalaxyParameters object
- gk.error : the error margin of above galaxy parameters
- gk.true_flux_map : the intrinsic flux map
- gk.true_velocity_map : the intrinsic velocity field
- gk.true_disp_map : the intrinsic dispersion map

Save the run

Once the MCMC has run, you can save the results to file easily :

It will create a bunch of files prefixed by my_run in the current working directory.

```
In [36]: gk = galpak.run(path,instrument=galpak.MUSEWFM())
         gk.save('my_run')
```

```
INFO:GalPaK:Setting 95 percentiles
INFO:GalPaK:self.galaxy : fittest parameters :
 Galaxy Parameters : value (± stdev) [units]     [Confidence Interval]
    x: 14.56 ± 0.10 (pixel)      CI 95%: [14.36,14.79]
    y: 14.58 ± 0.07 (pixel)      CI 95%: [14.44,14.71]
    z: 14.566303 ± 0.064328 (pixel)      CI 95%: [14.439964,14.688898]
    flux: 8.54e-17 ± 2.37e-18 (same units as the input Cube)     CI 95%: [8.02e-17,9.07e-17]
    radius: 4.06 ± 0.24 (pixel)  CI 95%: [3.63,4.68]
    inclination: 68.77 ± 2.37 (deg)      CI 95%: [64.74,73.76]
    pa: 126.11 ± 2.00 (deg)      CI 95%: [122.16,130.26]
    turnover_radius: 1.34 ± 0.21 (pixel)         CI 95%: [0.89,1.62]
    maximum_velocity: 194.91 ± 12.63 (km/s)      CI 95%: [178.07,221.67]
    velocity_dispersion: 83.66 ± 5.51 (km/s)     CI 95%: [73.45,92.68]
```

Level 0

Running the program, default parameters and save files

Level 1

Tuning, custom boundaries, plotting and recover things

MCMC chain tuning

Again, you can see the detailed documentation for the available parameters, but the most important parameters to tune are :

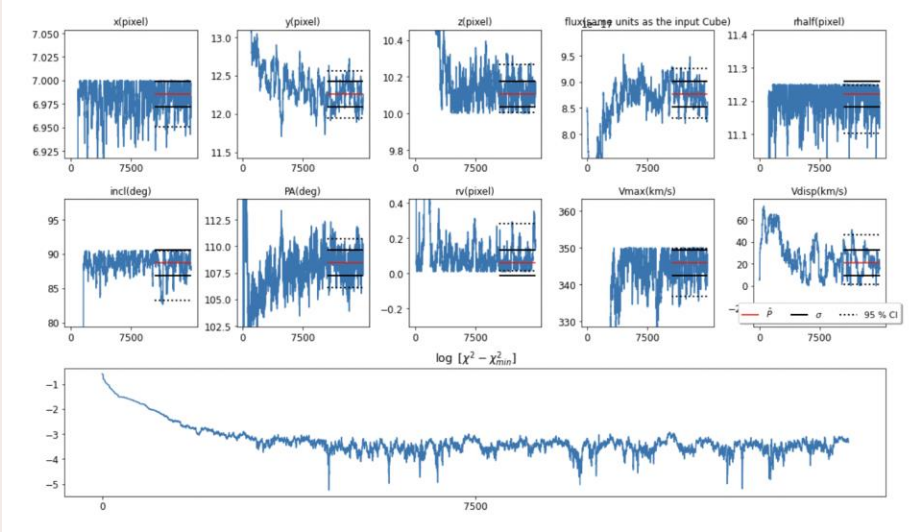
- **random_scale :** a scale factor to the width of the proposal distribution (Cauchy). A good practice is to tune this to have an acceptance rate of 30-50 %. (for example, the value random_scale=2 sets a factor 2x from the defaults)
- **max_iterations :** max number of (accepted) iterations **(default=10000)**
- **method :** 'last' or 'chi_sorted' or 'chi_min' Method used to determine the best parameters from the chain.
  - 'last' (default) : mean of the last_chain_fraction(%) last parameters of the chain
  - 'chi_sorted' : mean of the last_chain_fraction(%) best fit parameters of the chain
  - 'chi_min' : mean of last_chain_fraction(%) of the chain around the min chi
- **last_chain_fraction :** last fraction of chain (in %) to use to determine the best parameters (default=60)
- **min_acceptance_rate :** minimum acceptance rate (in %) to keep going. **(default: 5)**

Using Custom Boundaries

You can make sure that GalPaK will not try to find galaxy parameters outside of explicit boundaries :

```
In [11]: from galpak import run, GalaxyParameters
         min_boundaries = GalaxyParameters(x=5.)
         max_boundaries = GalaxyParameters(x=7., y=9.)
         gk = run(path,
                  min_boundaries=min_boundaries,
                  max_boundaries=max_boundaries)
```

We're getting close to something good!!!



The boundaries you provide will be merged into the default boundaries.

Note: For Vmax ‘maximum_velocity’, the min and max boundaires should be equal, e.g (-300,300)

Plotting the Markov Chain

Once a deconvolution has been computed, you can plot the Markov chain that was created :

```
In [12]: glpk3d = GalPaK3D(path, seeing=1.0, instrument=galpak.MUSE(lsf_fwhm=2.51))
         galaxy = glpk3d.run_mcmc(max_iterations=15e3)

         # Show plot on-screen
         glpk3d.plot_mcmc()

         # Save plot to png or jpg file
         glpk3d.plot_mcmc(filepath='my_mcmc_plot.png')
```

Plotting the Markov Chain

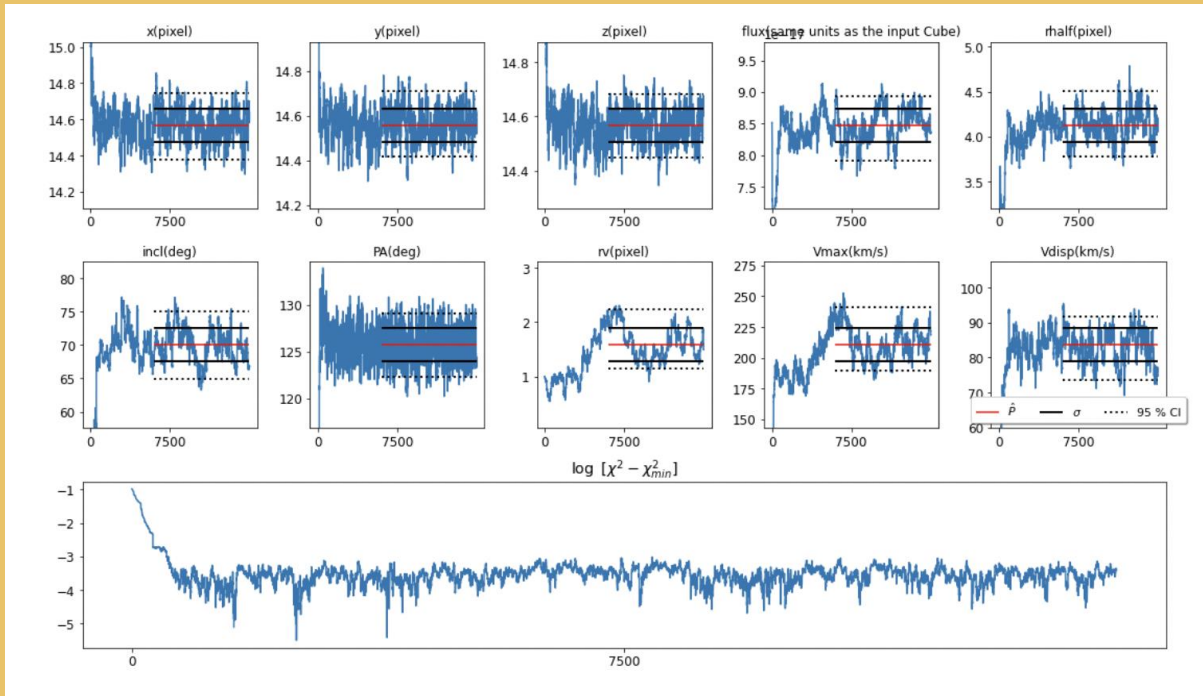
Once a deconvolution has been computed, you can plot the Markov chain that was created :

```
In [12]: glpk3d = GalPaK3D(path, seeing=1.0, instrument=galpak.MUSE(lsf_fwhm=2.51))
         galaxy = glpk3d.run_mcmc(max_iterations=15e3)

         # Show plot on-screen
         glpk3d.plot_mcmc()

         # Save plot to png or jpg file
         glpk3d.plot_mcmc(filepath='my_mcmc_plot.png')
```

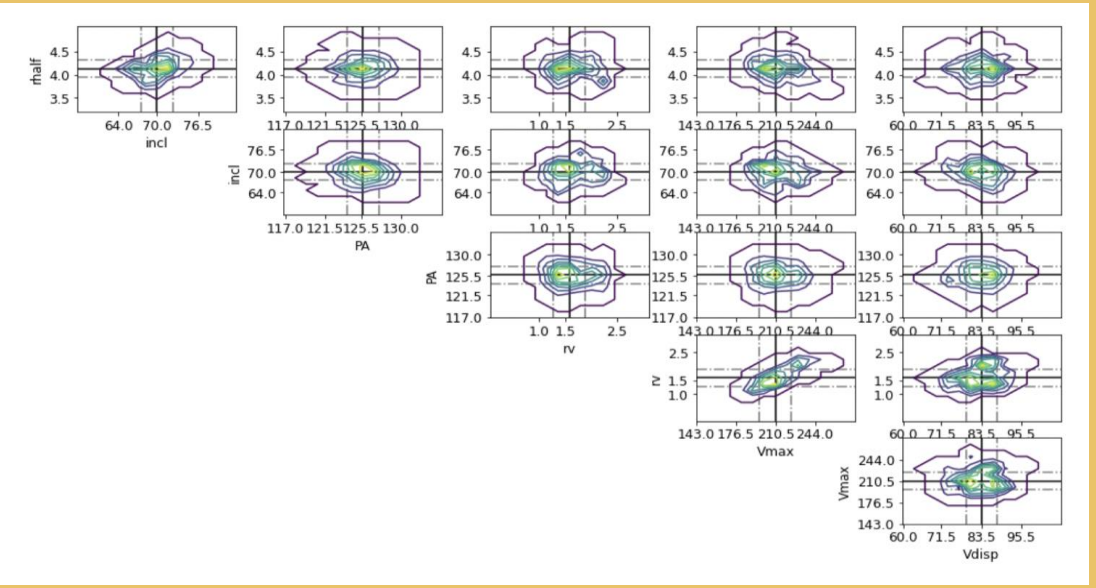
YEAAAH!!!



Plotting the cross-correlations

After a galpak run has been completed, you can plot the correlations in the Markov chain with :

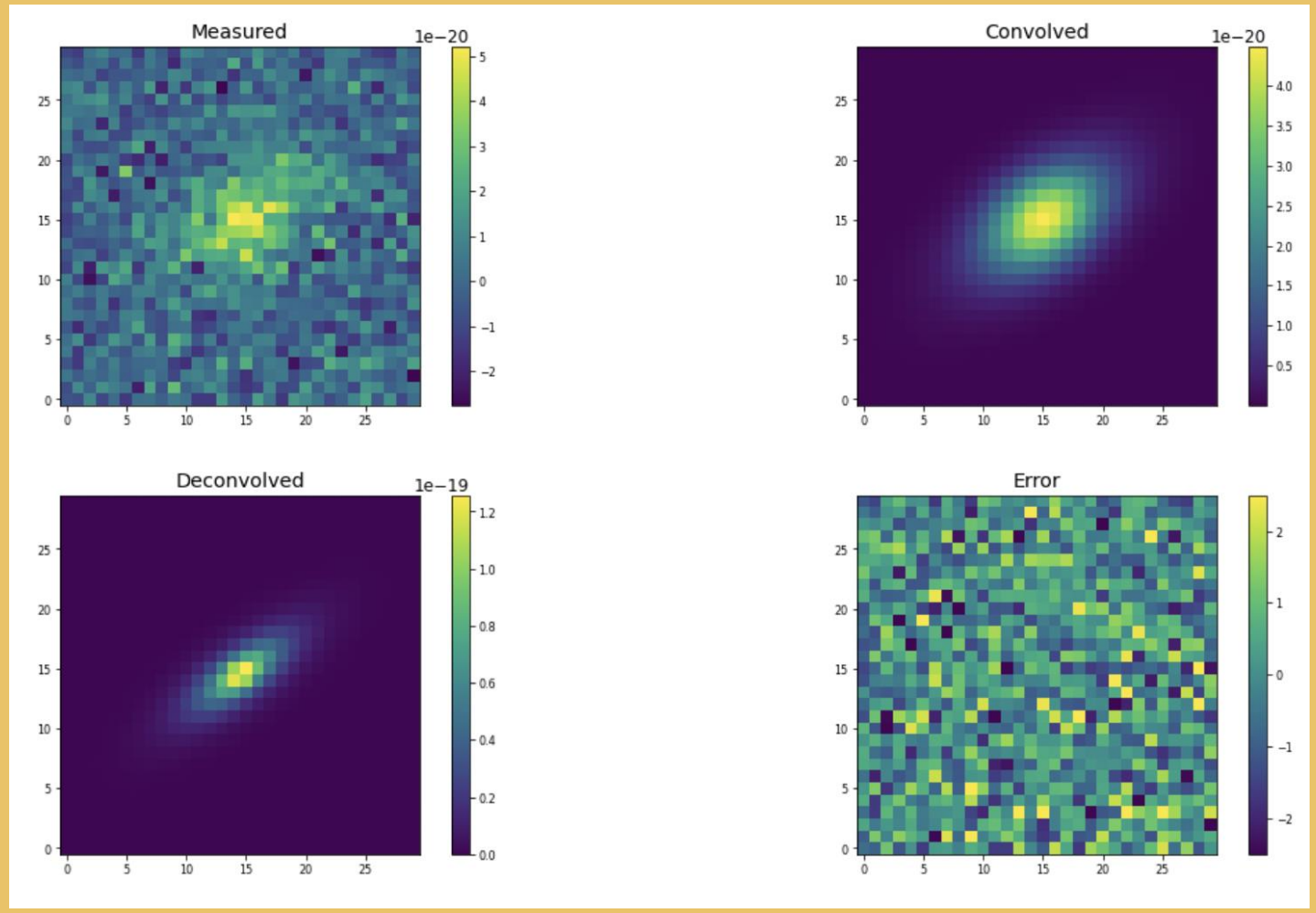
```
In [20]: # Save plot to png or jpg file
         glpk3d.plot_correlations(filepath='my_correlations.png')
```



Plotting the cubes' images

Once a deconvolution has been computed, you can plot the resulting cubes :

```
In [17]: # Save plot to png or jpg file
         glpk3d.plot_images(filepath='my_plot.png')
```



Recover the parameters from an earlier save

Once you've saved the run to disk, you can use the following from_file method to read the parameters:

In [30]:
```
import asciitable

with open('my_galpak_run_galaxy_parameters.dat', 'r') as param_file:
    param_data = asciitable.read(param_file.read(), Reader=asciitable.FixedWidth)

param_data
```

```
rec.array([(14.52903316, 14.57705672, 14.61667915, 7.17599946e-17, 3.11226781, 45.81665437, 129.14350173, 0.13824896, 177.175
84491, 40.34677046),
           ( 0.09548216,  0.1040362 ,  0.06746581, 3.56484659e-18, 0.2185596 ,  3.52886667,   5.21010809, 0.06444591,  21.353
49963,  9.94527722),
           (14.41701884, 14.36202024, 14.48277857, 6.73464046e-17, 2.85268247, 36.62276208, 124.07775555, 0.02694582, 136.727
23602, 29.70761933),
           (14.72452279, 14.79898953, 14.70676055, 7.78924823e-17, 3.48652191, 47.91020476, 146.19442745, 0.22644488, 193.984
82216, 57.1689526 )],
          dtype=[('x', '<f8'), ('y', '<f8'), ('z', '<f8'), ('flux', '<f8'), ('radius', '<f8'), ('inclination', '<f8'), ('pa',
'<f8'), ('turnover_radius', '<f8'), ('maximum_velocity', '<f8'), ('velocity_dispersion', '<f8')])
```

Recover the chain from an earlier save

Once you've saved the run to disk, you can use the following snippet to re-iterate through the chain:

```
In [23]: with open('my_galpak_run_chain.dat', 'r') as chain_file:
             data = asciitable.read(chain_file.read(), Reader=asciitable.FixedWidth)
         for data_record in data:
             print(data_record.x)
             print(data_record.reduced_chi)
```

```
14.99917893521753
1.2407997249881073
14.997340264478542
1.2401591127596767
14.999938436541955
1.239836880787652
14.97463858651654
1.238970915083983
14.976362852637497
1.2387500230703008
14.975412870633912
1.2386470014806963
14.97291959861061
1.2382489785135655
14.986925494325863
1.2377733379147777
14.985405358179394
1.237332443935064
14.986700185116778
1.2369544540608406
14.981199364134264
1.2368943203309615
14.962634817602574
```

```
In [24]: data
```

```
rec.array([(14.99917894, 15.00370987, 14.98240802, 8.50980753e-17, 3.00703162, 30.0199502 ,  26.87518145, 0.99946949, 100.9
5876613,  4.99175792, 1.24079972),
           (14.99734026, 15.13369208, 14.98650075, 8.51319683e-17, 3.00337415, 30.02336345,  27.11382556, 1.00610681, 104.3
647911 ,  4.94552949, 1.24015911),
           (14.99993844, 15.12907903, 14.9839588 , 8.50777496e-17, 3.00377415, 29.98219789,  27.17620683, 1.00424886, 104.6
5854313,  4.78635054, 1.23983688),
           (14.97463859, 15.12848799, 14.98016874, 8.48379575e-17, 3.00498666, 29.92502443,  27.05750314, 0.99760268, 104.7
7678733,  4.797414  , 1.23897092),
           (14.97636285, 15.11004916, 14.97611927, 8.47605313e-17, 2.99587162, 29.92692504,  27.530647  , 0.99808286, 105.1
8197385,  4.96585027, 1.23875002),
           (14.97541287, 15.11221483, 14.97940026, 8.47278690e-17, 2.9950459 , 29.78105349,  27.46318281, 0.99692437, 105.2
8924416,  5.34959853, 1.238647  ),
           (14.9729196 , 15.11303468, 14.9732625 , 8.46084305e-17, 2.99571594, 29.79429118,  27.40069571, 0.99812644, 105.5
1581136,  5.3060853 , 1.23824898),
           (14.98692549, 15.11267583, 14.97294822, 8.45746648e-17, 3.00544127, 29.78937069,  27.33450805, 0.98771642, 105.0
6131691,  5.47573807, 1.23777334),
           (14.98540536, 15.12054017, 14.96576706, 8.45969524e-17, 3.02110813, 29.79375213,  26.98970591, 0.98352259, 104.8
6168555,  5.55837022, 1.23733244),
           (14.98670019, 15.11856877, 15.00807154, 8.45265453e-17, 3.02528038, 29.80690651,  29.06907296, 0.98555042, 105.0
4213861,  1.72962769, 1.23695445),
           (14.98119936, 15.30058713, 15.01897224, 8.41493590e-17, 3.02923505, 29.74673225,  28.98604442, 0.98719082, 105.0
5600905,  1.75846731, 1.23689432),
           (14.96263482, 15.29745267, 15.01049743, 8.41484995e-17, 3.02975367, 29.74055226,  28.77634278, 0.9856127 , 105.0
```

Recover the chain from an earlier save

or simply using the import_chain method :

```
In [ ]: glpk3d = GalPaK3D(path)
        glpk3d.import_chain('my_galpak_run_chain.dat')
```

Cubes from another instrument

You can specify the instrument that will convolve the simulated data.

```
In [ ]: import galpak
        gk = galpak.run(new_path, instrument=galpak.MUSENFM())
```

Instruments accept two parameters: psf and lsf :

```
In [ ]: from galpak import run, MUSENFM, GaussianPointSpreadFunction, GaussianLineSpreadFunction
        my_psf = GaussianPointSpreadFunction(fwhm=1.2, pa=0., ba=1.0)
        my_lsf = GaussianLineSpreadFunction(fwhm=0.00065)
        gk = run(new_path, instrument=MUSENFM(psf=my_psf, lsf=my_lsf))
```

By default, the instrument will combine the psf (2D) and the lsf (1D) into a 3D spread function and will apply it to the cube in the method convolve(cube).

Level 1

Tuning, custom boundaries, plotting and recover things

Level 2

Customizing

Setting a parameter to a fixed value

You can use the parameter known_parameter, which takes a **GalaxyParameters** as in the example here:

```
In [26]: fixed = galpak.GalaxyParameters().copy()
         fixed.turnover_radius = 1
         gk = GalPaK3D(path, seeing=1.0, instrument=galpak.MUSE(lsf_fwhm=2.51))
         gk.run_mcmc(max_iterations=500,known_parameters=fixed)
```

```
    1 MIN=    1  0% χ²=1.251326>1.251326 x=15.00 y=14.99 z=15.003837 flux=8.51e-17 rhalf=3.00 incl=29.91 PA=307.26 rv=1.00
Vmax=99.94 Vdisp=5.13
    2 MIN=    2 25% χ²=1.250926>1.250926 x=14.99 y=15.00 z=15.011048 flux=8.50e-17 rhalf=3.00 incl=29.94 PA=307.76 rv=1.00
Vmax=99.79 Vdisp=5.18
    3 MIN=    3 13% χ²=1.249620>1.249620 x=15.01 y=15.00 z=15.008742 flux=8.50e-17 rhalf=3.04 incl=29.95 PA=307.75 rv=1.00
Vmax=99.88 Vdisp=5.02
    4 MIN=    4 17% χ²=1.249273>1.249273 x=15.01 y=15.00 z=15.007525 flux=8.50e-17 rhalf=3.04 incl=29.97 PA=307.86 rv=1.00
Vmax=100.51 Vdisp=4.71
    5 MIN=    5 21% χ²=1.249020>1.249020 x=15.00 y=15.00 z=15.010434 flux=8.51e-17 rhalf=3.05 incl=29.96 PA=307.56 rv=1.00
Vmax=101.51 Vdisp=4.71
    6 MIN=    6 24% χ²=1.247591>1.247591 x=14.99 y=15.03 z=14.968941 flux=8.51e-17 rhalf=3.08 incl=29.99 PA=308.40 rv=1.00
Vmax=101.74 Vdisp=4.56
    7 MIN=    7 25% χ²=1.246946>1.246946 x=14.99 y=15.02 z=14.973596 flux=8.50e-17 rhalf=3.10 incl=29.99 PA=308.34 rv=1.00
Vmax=101.42 Vdisp=4.44
    8 MIN=    8 28% χ²=1.246285>1.246285 x=14.98 y=15.03 z=14.963824 flux=8.50e-17 rhalf=3.11 incl=29.97 PA=308.32 rv=1.00
Vmax=101.34 Vdisp=4.54
    9 MIN=    9 29% χ²=1.242644>1.242644 x=14.98 y=15.03 z=14.959152 flux=8.50e-17 rhalf=3.13 incl=29.95 PA=351.85 rv=1.00
Vmax=100.99 Vdisp=4.70
   10 MIN=   10 30% χ²=1.242512>1.242512 x=15.00 y=15.02 z=14.957493 flux=8.49e-17 rhalf=3.13 incl=30.29 PA=351.95 rv=1.00
Vmax=100.96 Vdisp=4.70
   11 MIN=   11 31% χ²=1.242142>1.242142 x=14.99 y=15.02 z=14.912056 flux=8.47e-17 rhalf=3.13 incl=30.30 PA=351.82 rv=1.00
Vmax=100.83 Vdisp=5.26
```

Using line doublets ([OII] for example)

You can use the parameter line, a dictionary, to tell galpak you're expecting a dual peak :

```
In [27]: glpk3d = GalPaK3D(path, line={'wave': [3726.2, 3728.9], 'ratio': [0.8, 1.0]})
         galaxy = glpk3d.run_mcmc()
```

Other things!

- Customizing the Point Spread Function
- Customizing the Line Spread Function
- Creating a custom instrument
- Adding another Galaxy Parameter

Level 2

Customizing

Two examples where GalPaK is used!

Birkin, J. E., Puglisi, A., Swinbank, A. M., et al. 2024, **KAOSS: turbulent, but disc-like kinematics in dust-obscured star-forming galaxies at z ~1.3-2.6.** https://arxiv.org/abs/2301.05720

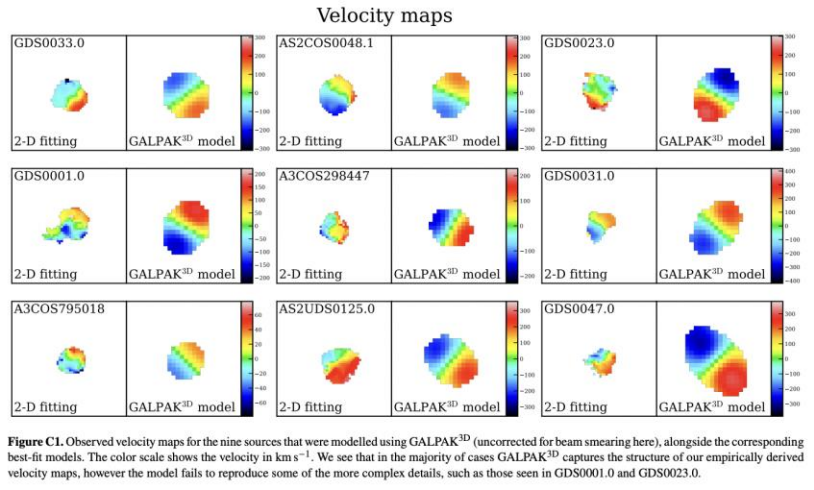


Figure C1. Observed velocity maps for the nine sources that were modelled using GALPAK^{3D} (uncorrected for beam smearing here), alongside the corresponding best-fit models. The color scale shows the velocity in km s^{-1}. We see that in the majority of cases GALPAK^{3D} captures the structure of our empirically derived velocity maps, however the model fails to reproduce some of the more complex details, such as those seen in GDS0001.0 and GDS0023.0.

Tejos, N., Lopez, S., Ledoux, C., et al. 2021, **Telltale signs of metal recycling in the circumgalactic medium of a z ~ 0.77 galaxy.** https://arxiv.org/pdf/2105.01673

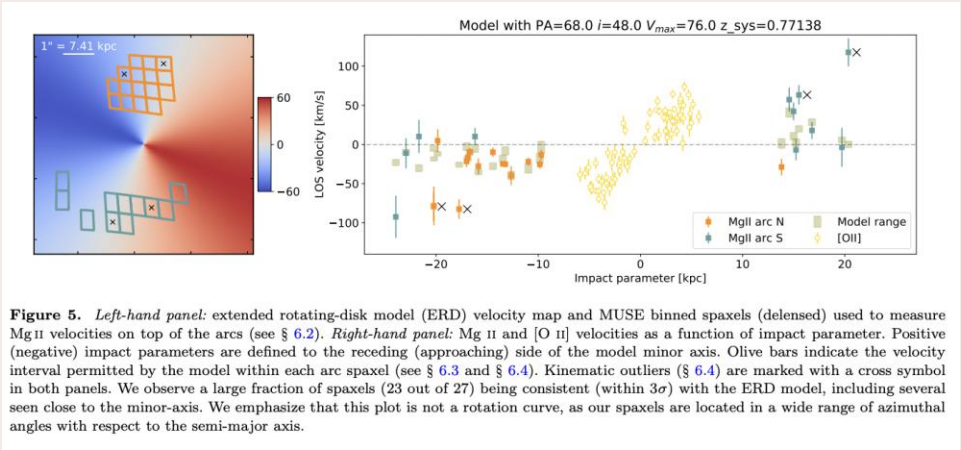


Figure 5. *Left-hand panel:* extended rotating-disk model (ERD) velocity map and MUSE binned spaxels (delensed) used to measure Mg II velocities on top of the arcs (see § 6.2). *Right-hand panel:* Mg II and [O II] velocities as a function of impact parameter. Positive (negative) impact parameters are defined to the receding (approaching) side of the model minor axis. Olive bars indicate the velocity interval permitted by the model within each arc spaxel (see § 6.3 and § 6.4). Kinematic outliers (§ 6.4) are marked with a cross symbol in both panels. We observe a large fraction of spaxels (23 out of 27) being consistent (within 3σ) with the ERD model, including several seen close to the minor-axis. We emphasize that this plot is not a rotation curve, as our spaxels are located in a wide range of azimuthal angles with respect to the semi-major axis.

3. Examples

Questions?