Twind

Release latest

Sep 22, 2020

User Guide

Twind is a Python prototype of TIGRESS Multiphase Wind Launching Model. The model is built on the TIGRESS simulation suite developed as part of the SMAUG project.

CHAPTER 1

Basic Usage

If you want to obtain scaling relations between wind (mass, momentum, energy, and metal) loading factors and star formation rate surface density at different escape velocity cuts, you would construct PDFs with **Twind**:

```
import twind

tw = twind.TigressWindModel(z0='H',verbose=True)
tw.set_axes(verbose=True)
pdf = tw.build_model(renormalize=True,energy_bias=True)
```

This will return 3D PDFs in the $(v_{\rm out}, c_s, \Sigma_{\rm SFR})$ space. A more complete example is available in the *Quickstart* tutorial.

`pdf` stores all information using xarray. Then, additional manipulations are easy. If you want to apply velocity cuts to get loading factors with a selected condition, you would do something like:

```
dbinsq = pdf.attrs['dlogcs']*pdf.attrs['dlogvout']
cdf_over_vB100 = pdf['Mpdf'].where(pdf['vBz']>100).sum(dim=['logcs','logvout'])*dbinsq
etaM_over_vB100 = pdf['etaM']*cdf_over_vB100
```

You can get a quick and dirty plot for the mass loading factor:

```
etaM_over_vB100.plot()
```

A more complete example is available in the *Loading Factors from 3D PDFs* tutorial.

1.1 Installation

Since **Twind** is a pure Python module, it should be pretty easy to install. In addition to widely used packages like scipy, numpy, matplotlib, you'll need xarray and astropy. Also, paper-figures needs seaborn, CMasher, and cmocean.

1.1.1 Github sources

pip or conda can be used to install `xarray` (see their installation instruction) and `astropy`. Simply, the following command would work.

```
conda install -c conda-forge xarray astropy
```

Then, you are ready to clone the repository.

```
git clone https://github.com/changgoo/Twind.git your-twind-path
```

Add path to the source directory using `sys.path`; e.g.,

```
import sys
sys.path.insert('your-twind-path')
```

1.1.2 Package managers

The recommended way to install the stable version of **Twind** is using pip

```
pip install -U twind
```

1.2 Joint PDF Model

Note: See Kim et al. (2020b) for details.

1.2.1 Cool outflow model

The cool outflow ($T < 2\times10^4$K) in the TIGRESS suite is well described by a model combining log-normal and generalized gamma distribution:

$$\tilde{f}_M^{\rm cool}(u,w) = A_c\left(\frac{v_{\rm out}}{v_{\rm out,0}}\right)^2 \exp\left[-\left(\frac{v_{\rm out}}{v_{\rm out,0}}\right)\right] \exp\left[-\frac{1}{2}\left(\frac{\ln(c_{\rm s}/c_{\rm s,0})}{\sigma}\right)^2\right]$$

where $A_c = (\ln 10)^2/(2\pi\sigma^2)^{1/2} = 2.12/\sigma$.

$$\frac{v_{\rm out,0}}{\rm km/s} = v0\left(\frac{\Sigma_{\rm SFR}}{M_\odot{\rm kpc}^{-2}{\rm yr}^{-1}}\right)^{0.23} + 3$$

At $|z| = H$, we adopt $v0 = 25$, $c_{s,0} = 6.7$km/s, and $\sigma = 0.1$. We found the same function form with $(v0, c_{s,0}) = (45, 7.5)$, (45,8.5), and (60,10) works reasonably well at $|z| = 2H$, 500pc, and 1kpc.

1.2.2 Hot outflow model

The hot outflow ($T > 5\times10^5$K) in the TIGRESS suite is well described by a model combining two generalized gamma distributions:

$$\tilde{f}_M^{\rm hot}(u,w) = A_h\left(\frac{v_{\mathcal{B},z}}{v_{\mathcal{B},0}}\right)^2 \exp\left[-\left(\frac{v_{\mathcal{B},z}}{v_{\mathcal{B},0}}\right)^4\right] \left(\frac{\mathcal{M}}{\mathcal{M}_0}\right)^3 \exp\left[-\left(\frac{\mathcal{M}}{\mathcal{M}_0}\right)\right]$$

where $A_h \equiv 2(\ln 10)^2/\Gamma(1/2) = 5.98$, $v_{\mathcal{B},z} \equiv (v_{\rm out}^2 + c_s^2)^{1/2}$, and $\mathcal{M} = v_{\rm out}/c_s$.

$$\frac{v_{\mathcal{B},0}}{10^3 {\rm km/s}} = 2.4 \left(\frac{\Sigma_{\rm SFR,0}^{1/2}}{2 + \Sigma_{\rm SFR,0}^{1/2}} \right) + 0.8$$

where $\Sigma_{\rm SFR,0} \equiv \Sigma_{\rm SFR}/(M_\odot {\rm kpc}^{-2} {\rm yr}^{-1})$

We adopt $\mathcal{M}_0 = 0.5$ irrespective of z.

None **Note:** This tutorial was generated from an IPython notebook that can be downloaded here.

1.3 Quickstart

1.3.1 Step 1: initialize Twind class

We are initilizing the *TigressWindModel* class with default parameters set to match the TIGRESS simulation suite results at $|z| = H$ (see *Simulation PDFs*). Possible options are `z0=['H', '2H', '500', '1000']`.

```
import twind

tw=twind.TigressWindModel(z0='H',verbose=True)
```

```
number of wind phase = 2
galactic parameter = sfr
reference height = H
cool_params
    A_v = 2.302585092994046
    p_v = 1
    d_v = 2
    A_cs = 9.185985478173896
    cs0 = 6.7
    sigma = 0.1
    vout0 = 25.0
hot_params
    A_vB = 5.196378098798331
    p_vB = 4
    d_vB = 2
    A_M = 1.151292546497023
    Mach0 = 0.5
    p_M = 1
    d_M = 3
params
    Esn = 1e+51 erg
    mstar = 95.5 solMass
    vcool = 200.0 km / s
    Mej = 10.0 solMass
    ZSN = 0.2
    ZISM0 = 0.02
    vej = 3171.4804794827423 km / s
ref_params
    Mref = 95.5 solMass
    pref = 2.5e+48 erg s / km
    Eref = 1e+51 erg
    Zref = 2.0 solMass
scaling_params
```

(continues on next page)

(continued from previous page)

```
    a = [-0.067 -1.216 -0.857  0.013 -1.426 -2.151 -1.013 -0.879 -2.228 -2.627
-0.698 -0.695  0.171 -0.942 -0.375  0.281]
    b = [-0.441 -0.227 -0.069 -0.418 -0.287 -0.149  0.016 -0.157 -0.117 -0.076
 0.136  0.108 -0.364 -0.146  0.04  -0.335]
    A = [0.86 0.06 0.14 1.03 0.04 0.01 0.1  0.13 0.01 0.   0.2  0.2  1.48 0.11
0.42 1.91]
    p = [0.559 0.773 0.931 0.582 0.713 0.851 1.016 0.843 0.883 0.924 1.136 1.108
0.636 0.854 1.04  0.665]
```

With the `verbose=True` option, key attributes are printed.

- `cool_params`: parameters for cool mass loading PDF. See *Joint PDF Model*
- `hot_params`: parameters for hot mass loading PDF. See *Joint PDF Model*
- `params`: other physical parameters related to the particular choices of the TIGRESS simulation suite (see Kim et al. (2020a))
- `ref_params`: outflow rates are normalized by $\Sigma_{\rm SFR} q_{\rm ref}/m_*$ to obtain loading factors
- `scaling_params`: fitting results for velocity-integrated loading factors as a function of SFR surface density (in log-log space) presented in Kim et al. (2020a). Each array contains the results for four loading factors (mass, momentum, energy, metal) of four phases (cool, intermediate, hot, whole). E.g., first four values are the results for the mass loading factor of cool, intermediate, hot, and whole gas.
- `a` is the intercept
- `b` is the slope
- `A` is 10^a
- `p` is $b+1$ for flux scalings.

$$\eta_q = A\Sigma_{\rm SFR}^b$$

Note: $\Sigma_{\rm SFR}$ is in $M_\odot {\rm kpc}^{-2}{\rm yr}^{-1}$ everywhere in this document.

Note: $u \equiv \log v_{\rm out}$ and $w \equiv \log c_s$ as defined in Kim et al. (2020b).

1.3.2 Step 2: setup axes

We use xarray extensibly for easier manipulation with broadcasting, indexing, slicing, and interpolation.

The *`TigressWindModel.set_axes()`* method accept either the simulated PDF (in the form of `xarray.Dataset`) or list of ranges (in log) and number of bins for `vout` and `cs` axes (`sfr` can either be a scalar or an array). Default is

- `vout` = (0,4,500)
- `cs` = (0,4,500)
- `sfr` = (-6,2,100)

This function will set attributes `u=logvout` and `w=logcs` as 1D `DataArray` as well as `vBz` and `Mach` as 2D `DataArray` for future use. If a range of `sfr` is passed, it will also set a member `logsfr` as 1D `DataArray` with different coordinates so that the final PDFs would be 3D `DataArray`.

For this example, we use a single value of SFR surface density and reduced number of bins for velocity axes.

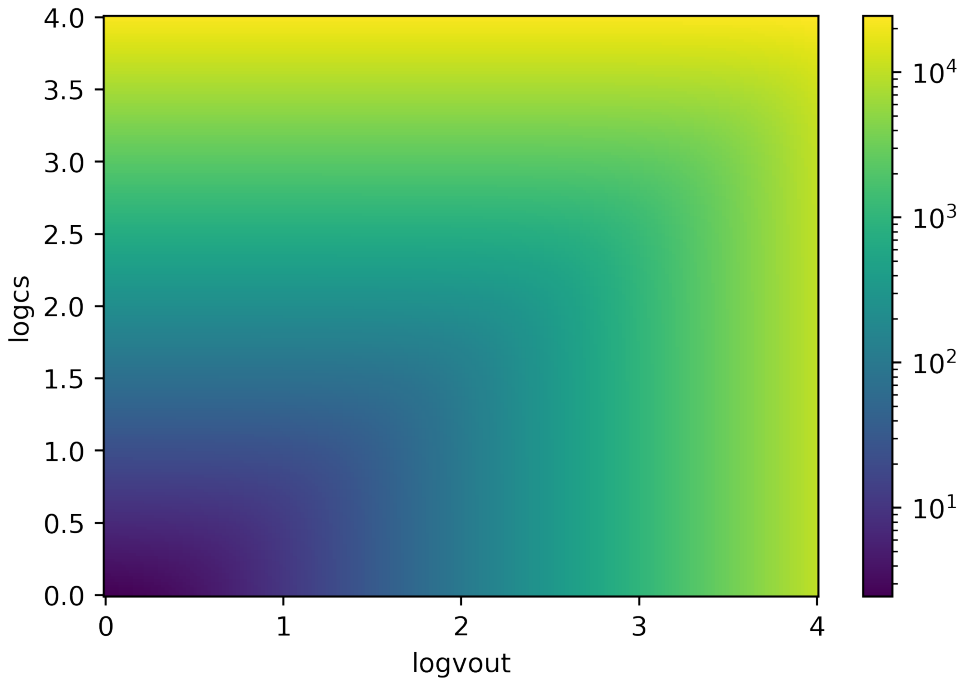
```
tw.set_axes(vout=(0,4,200),cs=(0,4,200),sfr=0.01,verbose=True)
```

```
sfr=0.01
cs: min=0, max=4, N=200
vout: min=0, max=4, N=200
```

We make sure that `vBz` and `Mach` are 2D while `u=logvout` and `w=logcs` are 1D.

```
print('u shpae:',tw.u.shape)
print('w shape:',tw.w.shape)
print('vBz shpae:',tw.vBz.shape)
print('Mach shape:',tw.Mach.shape)
g=tw.vBz.plot(norm=LogNorm())
```

```
u shpae: (200,)
w shape: (200,)
vBz shpae: (200, 200)
Mach shape: (200, 200)
```



1.3.3 Step 3: build mass loading PDFs

We have a method *`TigressWindModel.build_Mpdf()`* that automatically builds model PDFs for mass loading factor and return a `xarray.Dataset`. Note that if the range of `(u,w)` is not large enough, the mass PDF may not integrate to 1 (use `verbose=True` to check this).

Depending on the choice of the `sfr` axis, the resulting PDF can either be 2D or 3D. The returned `Dataset` have variables for PDFs (`Mpdf`, `Mpdf-cool`, `Mpdf-hot`) for `total`, `cool`, and `hot` outflow components. This also contains `vBz` and `Mach` as 2D arrays for convenience. In addition, the integrated loading factor (`etaM` and their

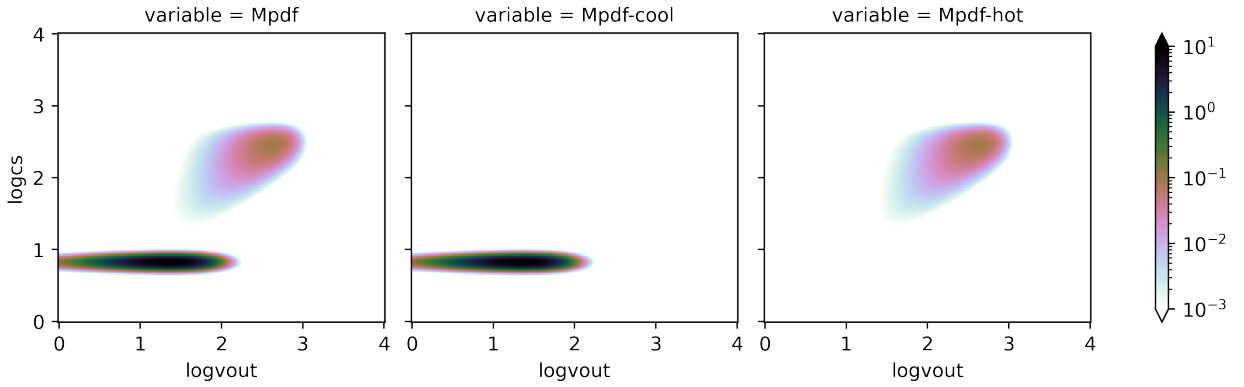
phase-separated values, i.e., `etaM-cool` and `etaM-hot`) as a function of `sfr` are saved. If `sfr` is a scalar, these are also scalars.

```
pdf = tw.build_Mpdf(verbose=True)
```

```
Mass PDFs are integrated to: cool=0.997 hot=1
```

```
pdf[['Mpdf','Mpdf-cool','Mpdf-hot']].to_array().plot(col='variable',
                                                     norm=LogNorm(vmin=1.e-3,vmax=10),
                                                     cmap=plt.cm.cubehelix_r
                                                    )
```

```
<xarray.plot.facetgrid.FacetGrid at 0x7f88b069a400>
```



1.3.4 Step 4: build all PDFs

We have a method *`TigressWindModel.build_model()`* that automatically builds model PDFs for mass, momentum, energy, and metal loading factors and return a `xarray.Dataset` containing all. The last three PDFs are reconstructed from the mass PDF as outlined in Kim et al. (2020b). By default, they are renormalized to ensure the integration over the entire `(u,w)` gives 1. Note that the metal PDF is not normalized for the input `ZISM` but for `ZISM0`.

Again, depending on the choice of the `sfr` axis, the resulting PDFs can either be 2D or 3D. The returned `Dataset` have variables for PDFs (`Mpdf`, `ppdf`, `Epdf`, `Zpdf`) and their phase-separated counterparts (e.g., `Mpdf-cool`, `Mpdf-hot`). The velocity-integrated loading factors (`etaM`, `etap`, `etaE`, `etaZ`) and their phase-separated counterparts (e.g., `etaM-cool` and `etaM-hot`) as a function of `sfr` are also stored. Finally, if `renormalize=True` (default), it also stores the renormalization factors (`p_renorm`, `E_renorm`, `Z_renorm`), which are also a function of `sfr`.

The `Dataset` has attributes for the choice of `ZISM` for the metal loading PDF as well as the bin sizes `dlogcs` and `dlogvout` for convenience.

```
pdf=tw.build_model(renormalize=True,energy_bias=True)
```

As it builds a model PDF, it automatically checks whether the mass PDFs are integrated to 1. I.e., both cool and hot PDFs should satisfy

$$\int\int \tilde{f}_M^{\rm ph} dudw = 1$$

individually. Again, this may depend on the `(u,w)` range. We then apply loading factor ratios to combine the mass loading PDF as

$$\tilde{f}_M = \frac{\eta_M^{\rm cool}}{\eta_M}\tilde{f}_M^{\rm cool} + \frac{\eta_M^{\rm hot}}{\eta_M}\tilde{f}_M^{\rm hot}$$

Note that `Mpdf-cool` and `Mpdf-hot` (and corresponding other PDFs) in the returned `Dataset` are not $\tilde{f}_M^{\rm ph}$ but $\frac{\eta_M^{\rm ph}}{\eta_M}\tilde{f}_M^{\rm ph}$.

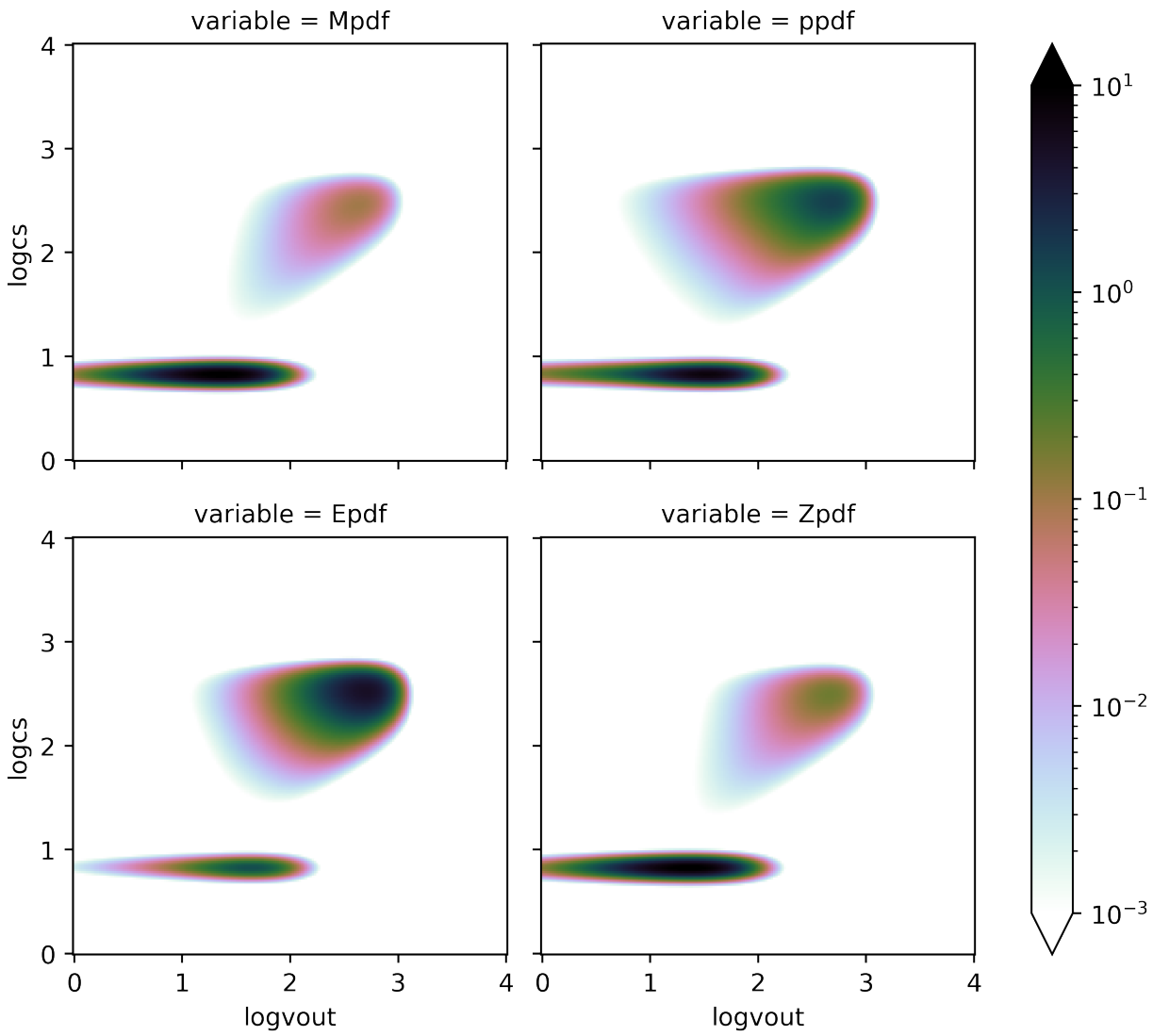
```
dudw=pdf.attrs['dlogvout']*pdf.attrs['dlogcs']
print('contribution to')
print('mass outflow rate from cool is {:.3f} and hot is {:.3f}'.format(
    pdf['Mpdf-cool'].sum().data*dudw,pdf['Mpdf-hot'].sum().data*dudw))
print('energy outflow rate from cool is {:.3f} and hot is {:.3f}'.format(
    pdf['Epdf-cool'].sum().data*dudw,pdf['Epdf-hot'].sum().data*dudw))
```

```
contribution to
mass outflow rate from cool is 0.968 and hot is 0.029
energy outflow rate from cool is 0.081 and hot is 0.919
```

Finally, 2D PDFs for mass, momentum, energy, and metal loadings at $\Sigma_{\rm SFR} = 10^{-2}$ look as follows.

```
pdf[['Mpdf','ppdf','Epdf','Zpdf']].to_array().plot(col='variable',col_wrap=2,
                                                   norm=LogNorm(vmin=1.e-3,vmax=10),
                                                   cmap=plt.cm.cubehelix_r
                                                  )
```

```
<xarray.plot.facetgrid.FacetGrid at 0x7f8870725898>
```



None **Note:** This tutorial was generated from an IPython notebook that can be downloaded here.

1.4 Loading Factors from 3D PDFs

See *Quickstart* for details of the first step. For this tutorial, we use the default ranges for axes setup.

```
import twind

tw=twind.TigressWindModel(z0='H',verbose=False)
tw.set_axes(verbose=True)
```

```
cs: min=0, max=4, N=500
vout: min=0, max=4, N=500
sfr: min=-6, max=2, N=100
```

Now we build joint PDFs as a function of SFR surface density, which gives a 3D PDF model.

```
pdf=tw.build_model(renormalize=True,energy_bias=True,verbose=False)
```

Note: After building the 3D PDF model, all subsequent manipulations heavily rely on the functionalities of xarray. We only show a few examples here.

1.4.1 Slicing back to 2D PDF

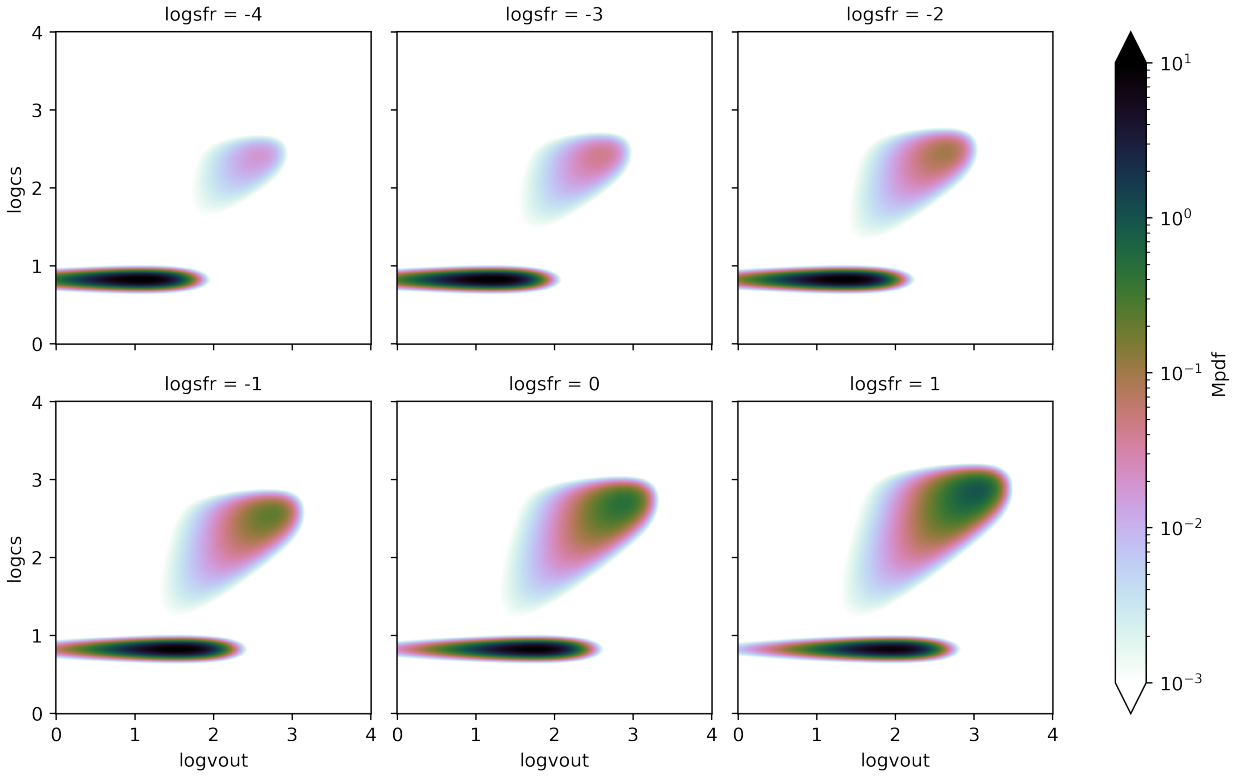
`xarray.Dataset` provides very useful functionality for slicing and interpolating through a method `sel`. Since we build a 3D PDF on the decretized `sfr` array, to get a PDF with a particular choice of `sfr` using the current 3D PDF, we would want either get the PDF with the closet value of `sfr` or interpolate to the chosen `sfr`. This can be done very easily with `xarray`.

```
# find the nearest one
pdf_nearest=pdf.sel(logsfr=-2, method='nearest')

# intepolate; this takes longer
pdf_interp=pdf.interp(logsfr=-2)

print('log sfr:{} {}'.format(pdf_nearest.logsfr.data,pdf_interp.logsfr.data))
```

```
log sfr:-2.04040404040404 -2
```

Interpolation (or slicing) within the range of `sfr` would be useful to compare pdfs at different SFRs quickly.

```
g=pdf['Mpdf'].interp(logsfr=[-4,-3,-2,-1,0,1]).plot(col='logsfr',col_wrap=3,
                                                    norm=LogNorm(vmin=1.e-3,vmax=10),
                                                    cmap=plt.cm.cubehelix_r
                                                   )
```



1.4.2 Selecting escapable outflows: constant velocity cut

For a practical use, we may want to select outflows with a certain velocity (or Bernoulli velocity) cut given a halo escape velocity. Here's how to do this.

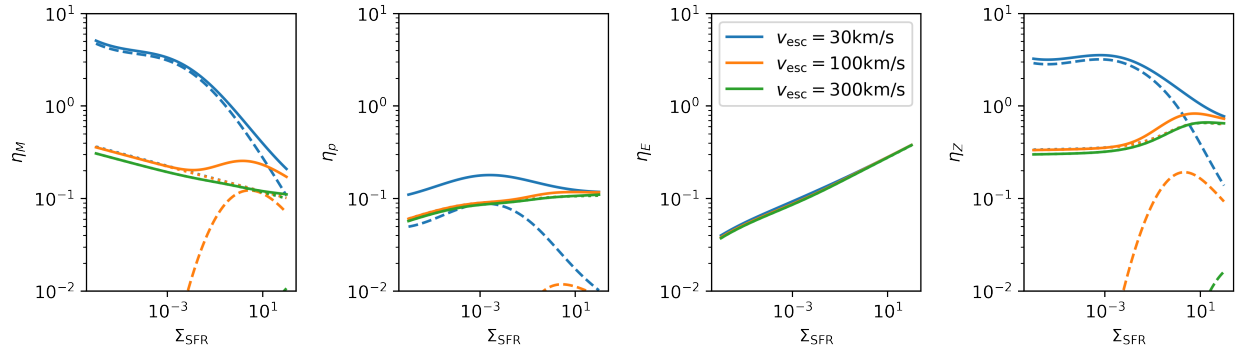
We may first want to calculate the cumulative distribution functions (CDF) with `vBz > vBzcut`. We then obtain the loading factors of selected gas by multiplying the total loading factors, which are stored as `etaM`, `etap`, `etaE`, `etaZ` in the original `Dataset`.

```
dbinsq=pdf.attrs['dlogcs']*pdf.attrs['dlogvout']
pdfs=['Mpdf','Mpdf-cool','Mpdf-hot',
      'ppdf','ppdf-cool','ppdf-hot',
      'Epdf','Epdf-cool','Epdf-hot',
      'Zpdf','Zpdf-cool','Zpdf-hot',]
sfr=10.**pdf['logsfr']

# For a constant velocity cut
fig, axes = plt.subplots(1,4,figsize=(10,3))
for vBzcut0 in [30,100,300]:
    cdf=pdf[pdfs].where(pdf['vBz']>vBzcut0).sum(dim=['logcs','logvout'])*dbinsq
    for ax,q in zip(axes.flat,['M','p','E','Z']):
        plt.sca(ax)
        eta=pdf['eta'+q]
        l,=plt.plot(sfr,cdf[q+'pdf']*eta,label=r'$v_{{\rm esc}}={}{{\rm km/s}}$'.
↪format(vBzcut0))
        plt.plot(sfr,cdf[q+'pdf-cool']*eta,ls='--',color=l.get_color())
        plt.plot(sfr,cdf[q+'pdf-hot']*eta,ls=':',color=l.get_color())
        plt.ylabel(r'$\eta_{}$'.format(q))
        plt.xlabel(r'$\Sigma_{\rm SFR}$')
```

(continues on next page)

(continued from previous page)

```
axes[2].legend()
plt.setp(axes,'xscale','log')
plt.setp(axes,'yscale','log')
plt.setp(axes,'ylim',(1.e-2,1.e1))
plt.tight_layout()
```



Note: $\Sigma_{\rm SFR}$ is in $M_\odot{\rm kpc}^{-2}{\rm yr}^{-1}$ everywhere in this document.

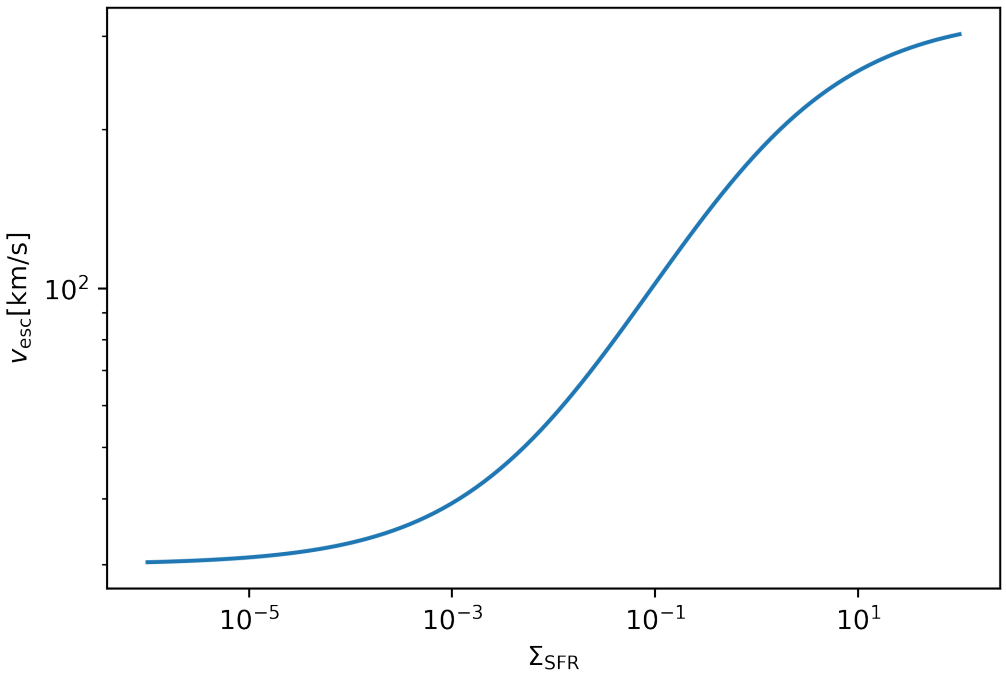
The mass (and metal) loading factor η_M changes dramatically as different velocity cuts are applied. This is because the cool outflow carries most of mass (shown as dashed lines), which has typical outflow velocities insufficient to escape for $v_{\rm esc} > 100{\rm km/s}$. However, at very high $\Sigma_{\rm SFR}$, cold outflow may contribute to the mass outflow rate significantly again, while one should take this with a grain of salt since those are outside our original parameter space $10^{-4} < \Sigma_{\rm SFR} < 1$ over which the model is calibrated. On the other hand, the energy loading factor η_E is unchaged since it is dominated by the hot outflow whose Bernoulli velocity is already larger than 300km/s even at low $\Sigma_{\rm SFR}$.

1.4.3 Selecting escapable outflows: SFR-dependent velocity cut

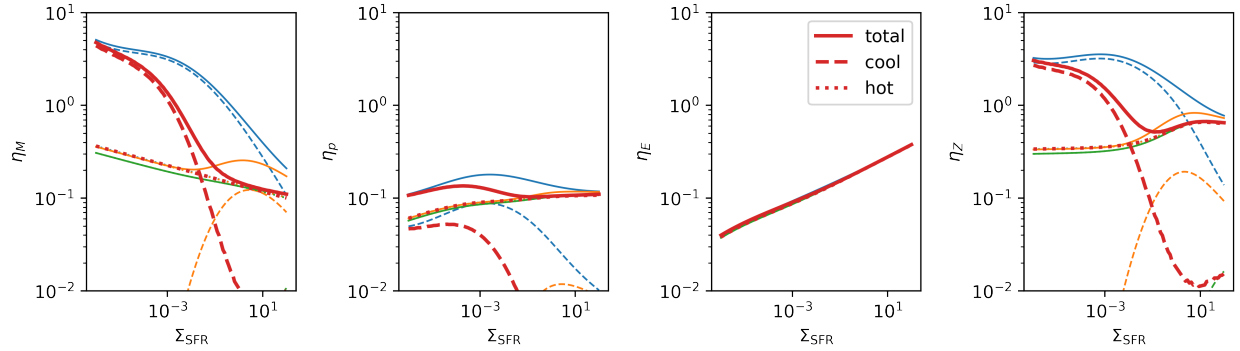
For some reasons, if the escape velocity is a function of SFR surface density, we get loading factors with a varying `vBzcut` easily using `xarray.Dataset.where` as it correctly broadcasts the information. Let's assume a particular form of the escape velocity:

$$v_{\rm esc} = 300{\rm km/s}\frac{\Sigma_{\rm SFR}^{1/2}}{\Sigma_{\rm SFR}^{1/2}+1} + 30{\rm km/s}$$

```
vBzcut = 300.*sfr**0.5/(sfr**0.5+1)+30
plt.loglog(sfr,vBzcut)
plt.xlabel(r'$\Sigma_{\rm SFR}$');
plt.ylabel(r'$v_{\rm esc} [{\rm km/s}]$');
```




```
fig, axes = plt.subplots(1,4,figsize=(10,3))

for vBzcut0 in [30,100,300]:
    cdf=pdf[pdfs].where(pdf['vBz']>vBzcut0).sum(dim=['logcs','logvout'])*dbinsq
    for ax,q in zip(axes.flat,['M','p','E','Z']):
        plt.sca(ax)
        eta=pdf['eta'+q]
        l,=plt.plot(sfr,cdf[q+'pdf']*eta,lw=1)
        plt.plot(sfr,cdf[q+'pdf-cool']*eta,ls='--',color=l.get_color(),lw=1)
        plt.plot(sfr,cdf[q+'pdf-hot']*eta,ls=':',color=l.get_color(),lw=1)
        plt.ylabel(r'$\eta_{}$'.format(q))
        plt.xlabel(r'$\Sigma_{\rm SFR}$')

cdf=pdf[pdfs].where(pdf['vBz']>vBzcut).sum(dim=['logcs','logvout'])*dbinsq
for ax,q in zip(axes.flat,['M','p','E','Z']):
    plt.sca(ax)
    eta=pdf['eta'+q]
    l,=plt.plot(sfr,cdf[q+'pdf']*eta,label='total',lw=2)
    plt.plot(sfr,cdf[q+'pdf-cool']*eta,ls='--',color=l.get_color(),label='cool',lw=2)
    plt.plot(sfr,cdf[q+'pdf-hot']*eta,ls=':',color=l.get_color(),label='hot',lw=2)
    plt.ylabel(r'$\eta_{}$'.format(q))
    plt.xlabel(r'$\Sigma_{\rm SFR}$')
axes[2].legend()

plt.setp(axes,'xscale','log')
plt.setp(axes,'yscale','log')
plt.setp(axes,'ylim',(1.e-2,1.e1))
plt.tight_layout()
```



Obviously, the result (red) falls between `vBzcut0=30` (blue) and `vBzcut0=300` (green) cases.

None **Note:** This tutorial was generated from an IPython notebook that can be downloaded here.

1.5 Particle Sampling

The *TigressWindSampler* class is a child class of the *TigressWindModel* class. For sampling, it won't build pdf models, but use the scaling relations and parameters of the model.

```
import twind

sampler=twind.TigressWindSampler()
```

1.5.1 Sampling at a single epoch

For this example, we use

- $\Sigma_{\rm SFR} = 10^{-2} M_\odot {\rm kpc}^{-2} {\rm yr}^{-1}$,
- area = $1 {\rm kpc}^2$
- dt = 1 Myr

and

- $m^{\rm cool} = m^{\rm hot} \in (10^2, 10^3) M_\odot$

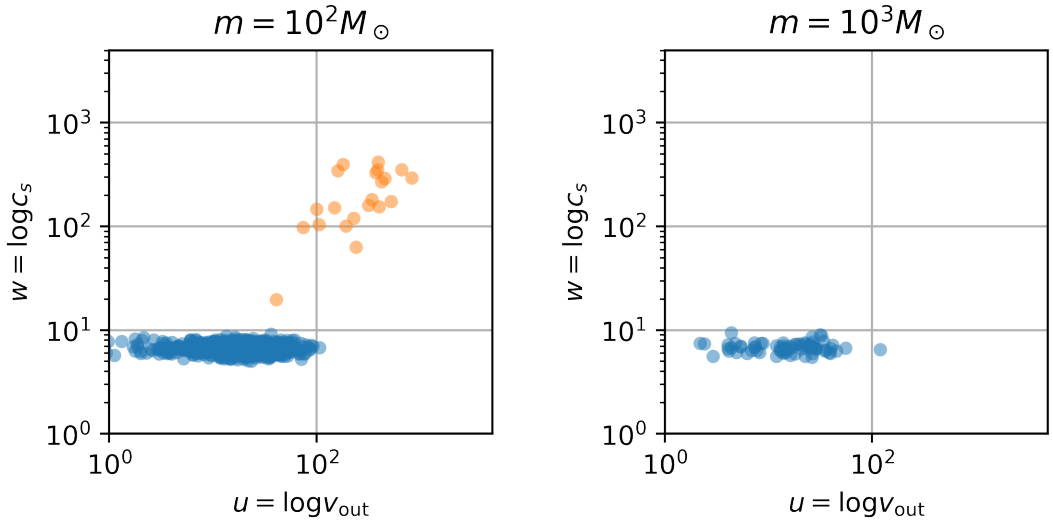
```
def draw_particle_dist(sfr0,area,dt,masses=[1.e2,1.e3,1.e4]):
    Nm=len(masses)
    fig, axes = plt.subplots(1,Nm,figsize=(3*Nm,3))

    for ax, m in zip(axes,masses):

        cool,hot=sampler.draw_mass(sfr0,m,m,area=area,dt=dt)

        plt.sca(ax)

        plt.plot(cool['vz'],cool['cs'],marker='o',ls='',ms=5,alpha=0.5,mew=0)
        plt.plot(hot['vz'],hot['cs'],marker='o',ls='',ms=5,alpha=0.5,mew=0)

        plt.xscale('log')
        plt.yscale('log')
        plt.title(r'$m = 10^{:.0f} M_\odot$'.format(np.log10(m)))
```

(continues on next page)

(continued from previous page)

```
        ax.grid('on')
        ax.set_aspect('equal')
        plt.xlim(1.,5.e3)
        plt.ylim(1.,5.e3)
        plt.xlabel(r'$u=\log v_{\rm out}$')
        plt.ylabel(r'$w=\log c_s$')
        plt.tight_layout()
    return fig
```

```
sfr0 = 1.e-2
area = 1
dt = 1.e6

f = draw_particle_dist(sfr0,area,dt,masses=[1.e2,1.e3])
```



The mass in the outflow can be obatined by

$$M_{\rm out} = \eta_M \Sigma_{\rm SFR} A dt$$

For a chosen $\Sigma_{\rm SFR}$, the total, cool, and hot mass loading factors are

```
etaM=sampler._eta_sfr_scaling(sfr0,'M_total')
etaMc=sampler._eta_sfr_scaling(sfr0,'M_cool')
etaMh=sampler._eta_sfr_scaling(sfr0,'M_hot')
print('eta_M={:.2f} eta_M^cool={:.2f} eta_M^hot={:.2f}'.format(etaM, etaMc, etaMh))
```

```
eta_M=7.07 eta_M^cool=6.54 eta_M^hot=0.19
```

which give the outflowing mass in each phase

```
print('M_out={:.3g} M_out^cool={:.3g} M_out^hot={:.3g}'.format(etaM*sfr0*area*dt,
↪etaMc*sfr0*area*dt,etaMh*sfr0*area*dt))
```

```
M_out=7.07e+04 M_out^cool=6.54e+04 M_out^hot=1.93e+03
```

Therefore, even for $m^{\rm hot} = 10^3 M_\odot$, we expect to sample a few particles as shown in the right panel of the above figure.

1.5.2 Sampling from a time series

For this example, we use a sinusoidal function for SFR surface density time series for 200 Myr with

- mean $\Sigma_{\rm SFR} = 10^{-3} M_\odot {\rm kpc}^{-2} {\rm yr}^{-1}$,
- period of 50 Myr

```
tmax = 2.e8
dt = 1.e6
time = np.arange(0,tmax,dt)
tp = 5.e7
sfr0 = 2.e-3
area = 1

sfr=sfr0*0.5*(np.sin(2*np.pi/tp*time)+2)
```

For a given time series of $\Sigma_{\rm SFR}$, we get reference values of outflow rates using the scaling relations of outflow loading factors (of each outflow phase) presented in Kim et al. (2020a).

The *TigressWindSampler.get_refs()* method returns four lists containing time series of reference outflow rates and loading factors for total, cool, and hot outflows. Each list contains `mass`, `momemtum`, `energy`, and `metal` in order.

```
refs,eta,etac,etah = sampler.get_refs(sfr)
```

```
mout = [eta[0]*refs[0]*area*dt, etac[0]*refs[0]*area*dt, etah[0]*refs[0]*area*dt]
Eout = [eta[2]*refs[2]*area*dt, etac[2]*refs[2]*area*dt, etah[2]*refs[2]*area*dt]
print('mean outflowing mass = {:.3g} (total) {:.3g} (cool)  {:.3g} (hot) Msun'.
↪format(mout[0].mean(),mout[1].mean(),mout[2].mean()))
print('mean outflowing energy = {:.3g} (total) {:.3g} (cool) {:.3g} (hot) erg'.
↪format(Eout[0].mean(),Eout[1].mean(),Eout[2].mean()))
```

```
mean outflowing mass = 2.73e+04 (total) 2.62e+04 (cool)  429 (hot) Msun
mean outflowing energy = 2.16e+51 (total) 4.3e+50 (cool) 1.82e+51 (hot) erg
```

For the area of 1 kpc^2 and time interval 1 Myr considered here, we expect the mean mass and energy in outflow are $2.7 \times 10^4 M_\odot$ and 2.2×10^{51} erg, respectively. The mass ratio between cool and hot outflows is about 50, therefore, for a fair sampling, we might need $m^{\rm cool}/m^{\rm hot} \sim 50$ with $m^{\rm cool} < 10^4 M_\odot$.

Frist, as a well sampled example, we use

- $m^{\rm cool} = 10^3 M_\odot$
- $m^{\rm hot} = 10^1 M_\odot$

```
def draw_particle_time_series(time, sfr, mc, mh, area, dt):
    refs,eta,etac,etah = sampler.get_refs(sfr)
    cool,hot=sampler.draw_mass(sfr,mc,mh,area=area,dt=dt)

    fig,axes = plt.subplots(4,1,sharex=True,figsize=(5,8))
    for p, etas_ in zip([cool,hot],[etac,etah]):
        outs=twind.to_time_series(p,time)

        for ax, q, qref, eta in zip(axes,outs,refs,etas_):
            plt.sca(ax)
            l,=plt.plot(time,q)
            plt.plot(time,eta*qref*area*dt,color=l.get_color(),ls='--')
            plt.yscale('log')
```

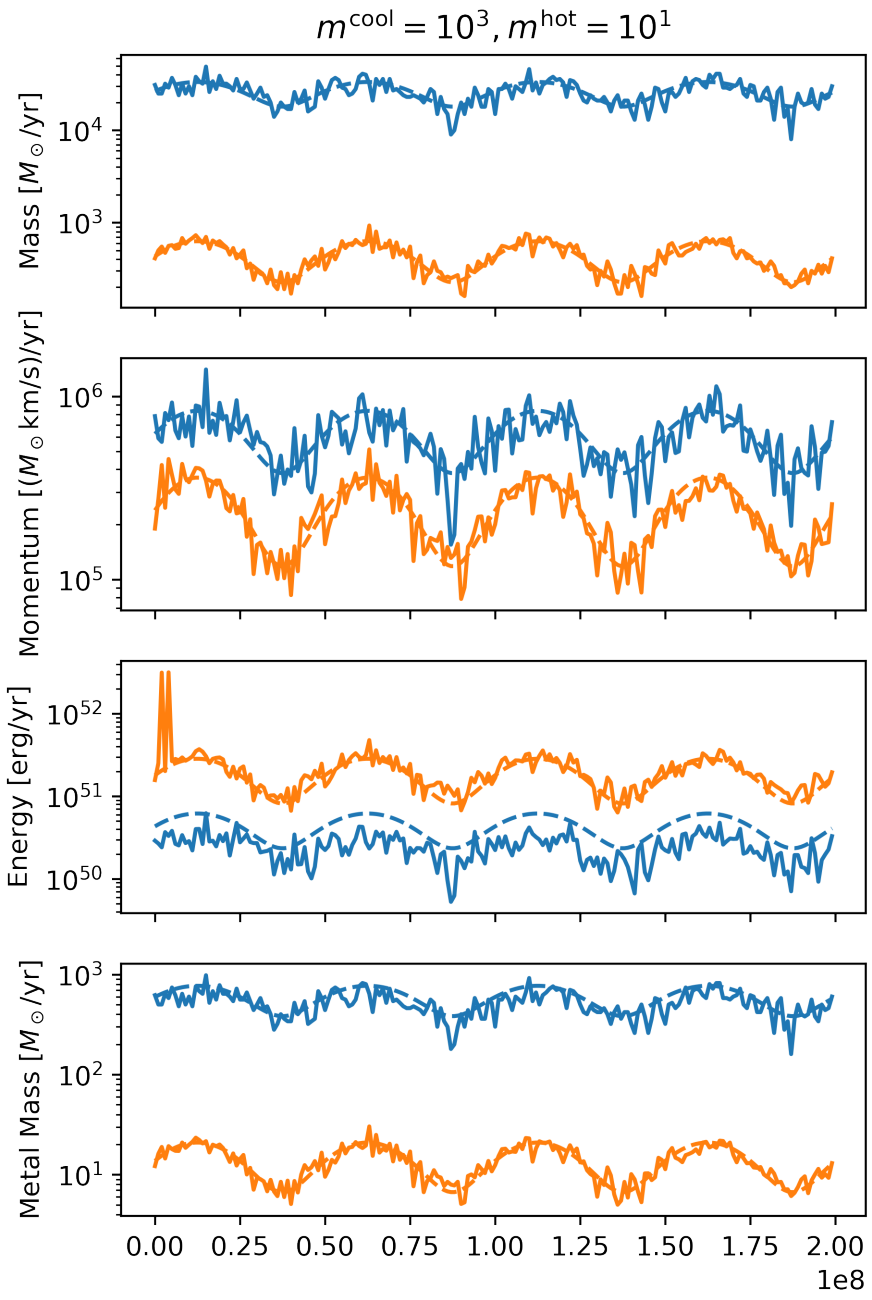
(continues on next page)

(continued from previous page)

```
    axes[0].set_title(r'$m^{{\rm cool}} = 10^{}, m^{{\rm hot}} = 10^{}$'.
↪format(int(np.log10(mc)),int(np.log10(mh))))
    axes[0].set_ylabel(r'Mass $[M_\odot/{\rm yr}]$')
    axes[1].set_ylabel(r'Momentum $[(M_\odot {\rm km/s})/{\rm yr}]$')
    axes[2].set_ylabel(r'Energy $[{\rm erg/yr}]$')
    axes[3].set_ylabel(r'Metal Mass $[M_\odot/{\rm yr}]$')
    return fig
```

Here, a utility function *to_time_series()* is used to convert particle data into time series of rates.

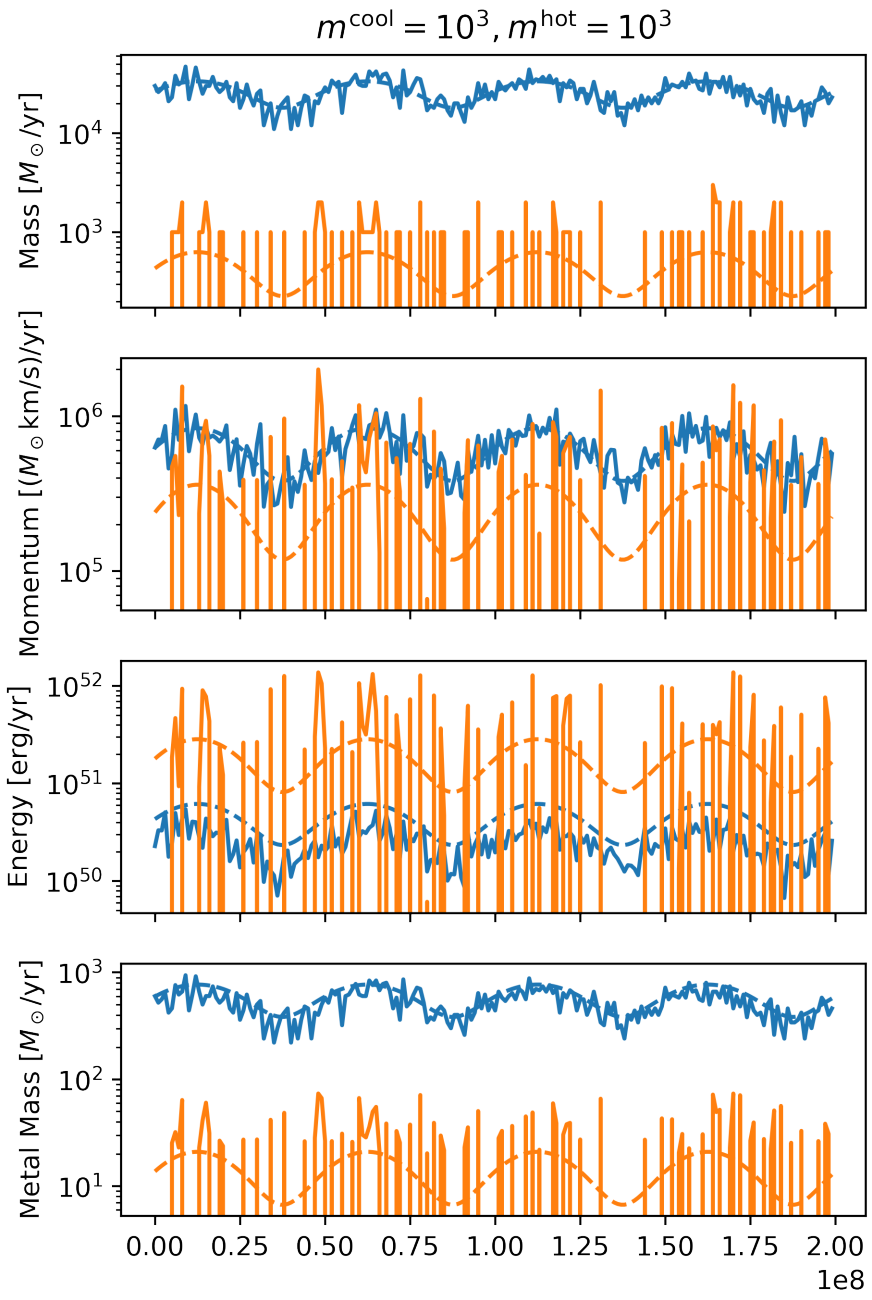
```
f = draw_particle_time_series(time, sfr, 1.e3, 1.e1, area, dt)
```



The second example is for well sampled cool phase but poorly sampled hot phase.

- $m^{\mathrm{cool}} = 10^3 M_\odot$
- $m^{\mathrm{hot}} = 10^3 M_\odot$

```
f = draw_particle_time_series(time, sfr, 1.e3, 1.e3, area, dt)
```



None **Note:** This tutorial was generated from an IPython notebook that can be downloaded here.

1.6 Simulation Model Tables

We construct model PDFs based on the results from the TIGRESS simulation suite presented in Paper I. We summarize model parameters and some integrated outflow propertes here using table files made available at zenodo or github. Mainly, the results are phase separated (three large bins in temperature or c_s) but outflow velocity integrated ($v_{\rm out} > 0$).

You can download the original notebook to reproduce tables and figures in Paper I.

1.6.1 Download and Prepare Tables

```
# Download Tables
import urllib.request
import os

repo_url='https://changgoo.github.io/tigress-wind-figureset'
tbl_files=['table-mean.ecsv','table-mean-err.ecsv']
if not os.path.isdir('tables/'): os.mkdir('tables/')
for f in tbl_files:
    if not os.path.isfile(f):
        urllib.request.urlretrieve('{}/tables/{}'.format(repo_url,f),'tables/'+f)
```

```
# Read Tables with astropy:

from astropy.table import QTable,Table
tmean=Table.read('tables/table-mean.ecsv')
terr=Table.read('tables/table-mean-err.ecsv')
```

```
# add additional time scales for Table 2 in Paper I
import astropy.constants as ac
import astropy.units as au
tmean['torb']=(2*np.pi/tmean['Omega_0'].quantity).to('Myr')
tmean['tosca']=(2*np.pi/np.sqrt(4*np.pi*ac.G*tmean['rho_tot'].quantity)).to('Myr')
tmean['toscn']=(2.0*np.pi*tmean['H'].quantity/tmean['sigma_eff'].quantity).to('Myr')


# set format for more compact display
for k in tmean.keys():
    if tmean[k].info.dtype == 'float64':
        tmean[k].info.format = '15.2g'
        if k in terr: terr[k].info.format = '15.2g'
```

1.6.2 Table 1: Model Parameters

```
table1_varlist=['model','Sigma_gas0','Sigma_star','rho_dm','Omega_0','z_star','R_0']
for k in table1_varlist:
    if tmean[k].info.dtype == 'float64':
        tmean[k].info.format = '15.3g'

tbl1=tmean[(tmean['z']=='H') & (tmean['phase']=='whole') ][table1_varlist]

tbl1.pprint_all()
```

```
model     Sigma_gas0       Sigma_star         rho_dm          Omega_0           z_star      ␣
↪       R_0
        solMass / pc2    solMass / pc2    solMass / pc3     km / (kpc s)          pc       ␣
↪       kpc
----- --------------- --------------- --------------- --------------- ---------------␣
↪---------------
   R2             150             450            0.08             100             245␣
↪              2
   R4              50             208           0.024            53.7             245␣
↪              4
```

(continues on next page)

(continued from previous page)

```
   R8              12              42          0.0064              28             245␣
↪               8
  R16            2.49            1.71         0.00143            11.9             245␣
↪              16
 LGR2             150             110           0.015              50             500␣
↪               2
 LGR4              60              50           0.005              30             500␣
↪               4
 LGR8              12              10          0.0016              15             500␣
↪               8
```

- **Sigma_gas0**: initial gas surface density, $\Sigma_{\rm gas,0}$
- **Sigma_star**: stellar surface density, Σ_*
- **rho_dm**: midplane dark matter density, $\rho_{\rm dm}$
- **Omega**: angular velocity of galactic rotation, Ω
- **R_0**: galactocentric radius, R_0
- **z_star**: scale height of stellar disk, z_*

1.6.3 Table 2: Time Scales

```
table2_varlist=['model','torb','toscn','tosca','tdep40','surf','sfr40']
tbl2=tmean[(tmean['z']=='H') & (tmean['phase']=='whole') ][table2_varlist]

tbl2.pprint_all()
```

```
model       torb           toscn           tosca           tdep40           surf      ␣
↪       sfr40
              Myr             Myr             Myr             Myr        solMass / pc2 ␣
↪solMass / (kpc2 yr)
----- --------------- --------------- --------------- --------------- ---------------␣
↪--------------------
   R2              61              32              23              66              74␣
↪                1.1
   R4         1.1e+02              51              38         2.4e+02              29␣
↪               0.12
   R8         2.2e+02         1.2e+02              75         2.1e+03              11␣
↪             0.0051
  R16         5.2e+02         4.6e+02         3.1e+02         3.1e+04             2.5␣
↪             8e-05
 LGR2         1.2e+02              52              48         1.5e+02              74␣
↪               0.49
 LGR4           2e+02              87              80         4.2e+02              38␣
↪               0.09
 LGR8         4.1e+02         2.2e+02         1.7e+02         3.3e+03              10␣
↪             0.0032
```

- **torb**: orbit time, $t_{\rm orb} = 2\pi/\Omega$
- **toscn**: vertical oscillation time derived from numerical measures, $t_{\rm osc,n} = 2\pi H/\sigma_{\rm z,eff}$
- **tosca**: vertical oscillation time derived from input parameters, $t_{\rm osc,a} = 2\pi/(4\pi G\rho_{\rm tot})^{1/2}$
- **tdep40**: gas depletion time with SFR surface density in 40 Myr, $t_{\rm dep,40} = \Sigma_{\rm gas}/\Sigma_{\rm SFR,40}$

- **surf**: mean gas surface density, $\Sigma_{\rm gas}$
- **sfr40**: mean SFR surface density from star particles young than 40 Myr, $\Sigma_{\rm SFR,40}$

mean and error are determined from bootstrap resampling with a sample size of 10 for time series over $0.5 < t/t_{\rm orb} < 1.5$

1.6.4 Table 3-1: Fluxes

```
z0='H' # height can be ('H','2H','500','1000')
table3_varlist1=['model','phase','mass','mom','energy','metal','metal_sn']
tbl3=tmean[tmean['z']==z0][table3_varlist1]

tbl3.pprint_all()
```

```
model phase         mass                   mom                 energy            metal␣
↪            metal_sn
            solMass / (kpc2 yr) km solMass / (kpc2 s yr) erg / (kpc2 yr) solMass /␣
↪(kpc2 yr) solMass / (kpc2 yr)
----- ----- ------------------- ------------------------ --------------- -------------
↪------ -------------------
   R2  cool                0.74                       50         7.2e+46
↪ 0.029              0.0032
   R2   int               0.063                       10         2.8e+46
↪0.0026             0.00056
   R2   hot                0.13                   1.4e+02         2.8e+48
↪0.0096              0.0062
   R2 whole                0.94                     2e+02         2.9e+48
↪ 0.041               0.01
   R4  cool                0.26                       12           1e+46
↪0.0081             0.00042
   R4   int               0.014                      1.8         4.1e+45
↪0.00047             7.1e-05
   R4   hot               0.026                       18         2.2e+47
↪0.0013             0.00058
   R4 whole                 0.3                       32         2.3e+47
↪0.0098              0.001
   R8  cool               0.032                     0.78         4.4e+44
↪0.00071             2.1e-05
   R8   int              0.0012                     0.12         2.3e+44
↪2.9e-05             2.9e-06
   R8   hot              0.0013                     0.67         5.5e+45
↪4.1e-05             1.5e-05
   R8 whole               0.035                      1.6         6.2e+45
↪0.00078             3.8e-05
  R16  cool              0.0055                    0.085         2.3e+43
↪0.00011             2.5e-09
  R16   int             3.6e-05                   0.0028         3.7e+42
↪7.7e-07             5.2e-08
  R16   hot             1.4e-05                   0.0093         6.1e+43
↪4.4e-07             1.8e-07
  R16 whole              0.0055                    0.097         8.8e+43
↪0.00011             8.4e-08
 LGR2  cool                0.55                       26         2.8e+46
↪ 0.018              0.0015
 LGR2   int               0.026                      3.6         8.8e+45
↪0.00097             0.00019
```

(continues on next page)

(continued from previous page)

```
 LGR2   hot               0.055                  48         6.8e+47
↪0.0033              0.0018
 LGR2 whole               0.63                  78         7.1e+47
↪ 0.023              0.0034
 LGR4  cool               0.45                  14         8.3e+45
↪ 0.012             0.00021
 LGR4   int               0.01                 1.2         2.5e+45
↪0.0003             3.7e-05
 LGR4   hot              0.015                  10         1.1e+47
↪0.00065             0.00028
 LGR4 whole               0.47                  25         1.2e+47
↪ 0.013             0.00048
 LGR8  cool               0.04                0.86         3.6e+44
↪0.00087              7.9e-06
 LGR8   int            0.00074               0.073         1.3e+44
↪1.7e-05             1.5e-06
 LGR8   hot            0.00089                0.44         3.3e+45
↪2.7e-05             8.6e-06
 LGR8 whole              0.042                 1.4         3.8e+45
↪0.00092             1.8e-05
```

- **mass**: mass flux, $\overline{\mathcal{F}}_M$
- **mom**: momentum flux, $\overline{\mathcal{F}}_p$
- **energy**: energy flux, $\overline{\mathcal{F}}_E$
- **metal**: metal flux, $\overline{\mathcal{F}}_Z$
- **metal_sn**: SN-origin metal flux, $\overline{\mathcal{F}}_Z^{SN}$

mean and error are determined from bootstrap resampling with a sample size of 10 for time series over $0.5 < t/t_{\rm orb} < 1.5$

1.6.5 Table 3-2: Loading Factors

```
z0='H' # height can be ('H','2H','500','1000')
table3_varlist2=['model','phase','mass_loading','mom_loading',
                 'energy_loading','metal_loading','metal_sn_loading',]
tbl3=tmean[tmean['z']==z0][table3_varlist2]

tbl3.pprint_all()
```

```
model phase   mass_loading    mom_loading    energy_loading  metal_loading  metal_sn_
↪loading

----- ----- --------------- --------------- --------------- --------------- ----------
↪------
   R2  cool            0.68           0.035          0.0064             1.3
↪  0.14
   R2   int           0.058          0.0071          0.0025            0.11
↪ 0.025
   R2   hot            0.12             0.1            0.24            0.42
↪  0.27
   R2 whole            0.86            0.14            0.25             1.8
↪  0.44
```

(continues on next page)

(continued from previous page)

```
   R4  cool             2.2          0.075          0.008           3.2
↪  0.17
   R4   int            0.12          0.012         0.0032          0.19
↪ 0.028
   R4   hot            0.22           0.12           0.17           0.5
↪  0.23
   R4 whole             2.5            0.2           0.18           3.9
↪   0.4
   R8  cool             6.3           0.12         0.0081           6.6
↪  0.19
   R8   int            0.24          0.018         0.0043          0.27
↪ 0.027
   R8   hot            0.25          0.099            0.1          0.38
↪  0.14
   R8 whole             6.8           0.23           0.11           7.3
↪  0.36
  R16  cool              56           0.66          0.022            54
↪0.0012
  R16   int            0.37          0.022         0.0036          0.38
↪ 0.025
  R16   hot            0.14          0.072           0.06          0.22
↪ 0.087
  R16 whole              56           0.75          0.086            55
↪ 0.041
 LGR2  cool             1.2          0.042         0.0056           1.9
↪  0.15
 LGR2   int           0.054         0.0058         0.0018         0.098
↪  0.02
 LGR2   hot            0.12          0.077           0.14          0.33
↪  0.18
 LGR2 whole             1.3           0.13           0.14           2.3
↪  0.35
 LGR4  cool               5           0.12         0.0088           6.2
↪  0.11
 LGR4   int            0.11           0.01         0.0027          0.16
↪  0.02
 LGR4   hot            0.17          0.085           0.11          0.35
↪  0.15
 LGR4 whole             5.3           0.21           0.12           6.8
↪  0.26
 LGR8  cool              12            0.2          0.011            13
↪  0.12
 LGR8   int            0.23          0.017          0.004          0.26
↪ 0.022
 LGR8   hot            0.28            0.1          0.099           0.4
↪  0.13
 LGR8 whole              13           0.32           0.11            14
↪  0.27
```

- **mass_loading**: mass loading factor, η_M
- **mom_loading**: mom loading factor, η_p
- **energy_loading**: energy loading factor, η_E
- **metal_loading**: mass loading factor, η_Z
- **metal_sn_loading**: SN-origin metal loading factor, η_Z^{SN}

mean and error are determined from bootstrap resampling with a sample size of 10 for time series over $0.5 < t/t_{\rm orb} < 1.5$

1.6.6 Table 4: Velocities and Metals

```
z0='H' # height can be ('H','2H','500','1000')

table4_varlist=['model','phase','vout_flux','vB','Z','enrichment','fmass_sn','fmetal_
↪sn']
tbl4=tmean[tmean['z']==z0][table4_varlist]

tbl4.pprint_all()
```

```
model phase    vout_flux           vB              Z           enrichment        fmass_
↪sn       fmetal_sn
                 km / s          km / s
----- ----- --------------- --------------- --------------- --------------- ----------
↪----- ----------------
   R2  cool              69           1e+02           0.039             1.1          ␣
↪0.026            0.14
   R2   int          1.4e+02         2.1e+02           0.042             1.2          ␣
↪0.044            0.21
   R2   hot          5.8e+02         1.4e+03           0.072             2.1          ␣
↪ 0.23            0.63
   R2 whole          1.6e+02         5.6e+02           0.044             1.3          ␣
↪0.059            0.27
   R4  cool              47              67           0.032             1.1          ␣
↪0.011           0.068
   R4   int          1.1e+02         1.6e+02           0.034             1.1          ␣
↪0.023            0.13
   R4   hot          3.8e+02         8.2e+02           0.046             1.6          ␣
↪0.095             0.4
   R4 whole            1e+02         3.2e+02           0.034             1.1          ␣
↪0.024            0.14
   R8  cool              20              37           0.022               1          ␣
↪0.0035           0.032
   R8   int              69         1.3e+02           0.024             1.1          ␣
↪0.012             0.1
   R8   hot          2.4e+02           6e+02           0.031             1.4          ␣
↪0.054            0.34
   R8 whole              34         1.4e+02           0.023             1.1          ␣
↪0.0066           0.057
  R16  cool             7.9              20            0.02               1          7.
↪3e-06         7.9e-05
  R16   int              36              95           0.022             1.1          ␣
↪0.0063           0.071
  R16   hot          1.3e+02         5.5e+02           0.032             1.6          ␣
↪0.051            0.37
  R16 whole             8.4              32            0.02               1          6.
↪3e-05         0.00068
 LGR2  cool              44              68           0.035             1.1          ␣
↪0.015           0.084
 LGR2   int          1.1e+02         1.8e+02           0.039             1.2          ␣
↪0.036            0.19
 LGR2   hot          4.2e+02           1e+03           0.057             1.8          ␣
↪ 0.15            0.51
```

(continues on next page)

(continued from previous page)

```
 LGR2 whole              92          3.4e+02            0.038             1.2
↪0.031            0.16
 LGR4  cool              30               45            0.028               1
↪0.0046           0.032
 LGR4   int              92          1.5e+02             0.03             1.1
↪0.018            0.12
 LGR4   hot          3.1e+02          7.4e+02            0.041             1.5
↪ 0.08            0.38
 LGR4 whole              47          1.7e+02            0.028             1.1
↪0.0085           0.058
 LGR8  cool              13               26            0.022               1
↪0.0014           0.013
 LGR8   int              50          1.2e+02            0.024             1.1
↪0.014            0.11
 LGR8   hot          1.6e+02          4.6e+02            0.029             1.4
↪0.039            0.29
 LGR8 whole              17               72            0.022               1
↪0.0032           0.027
```

- **vout_flux**: characteristic outflow velocity, $\overline{v}_{\rm out}$
- **vB**: Bernoulli velocity, $\overline{v}_{\mathcal{B}}$
- **Z**: outflow metallicity, $\overline{Z}$
- **enrichment**: metal enrichment factor, ζ
- **fmass_sn**: fraction of SN-origin mass flux, f_M^{SN}
- **fmetal_sn**: fraction of SN-origin metal flux, f_Z^{SN}

mean and error are determined from bootstrap resampling with a sample size of 10 for time series over $0.5 < t/t_{\rm orb} < 1.5$

None **Note:** This tutorial was generated from an IPython notebook that can be downloaded here.

1.7 Simulation PDFs

The time averaged joint PDFs from the TIGRESS simulations suites for all 7 models (see *Simulation Model Tables*) at all 4 heights are available. This tutorial demonstrates how to download, read, and handle simulation PDFs using `TigressSimLoader` class. More comphrensive examples can be found at *Figures in Paper II*.

```
import twind
```

```
# read in simulated PDF
sim = twind.TigressSimLoader('R4','H')
sim.load(download=True)
```

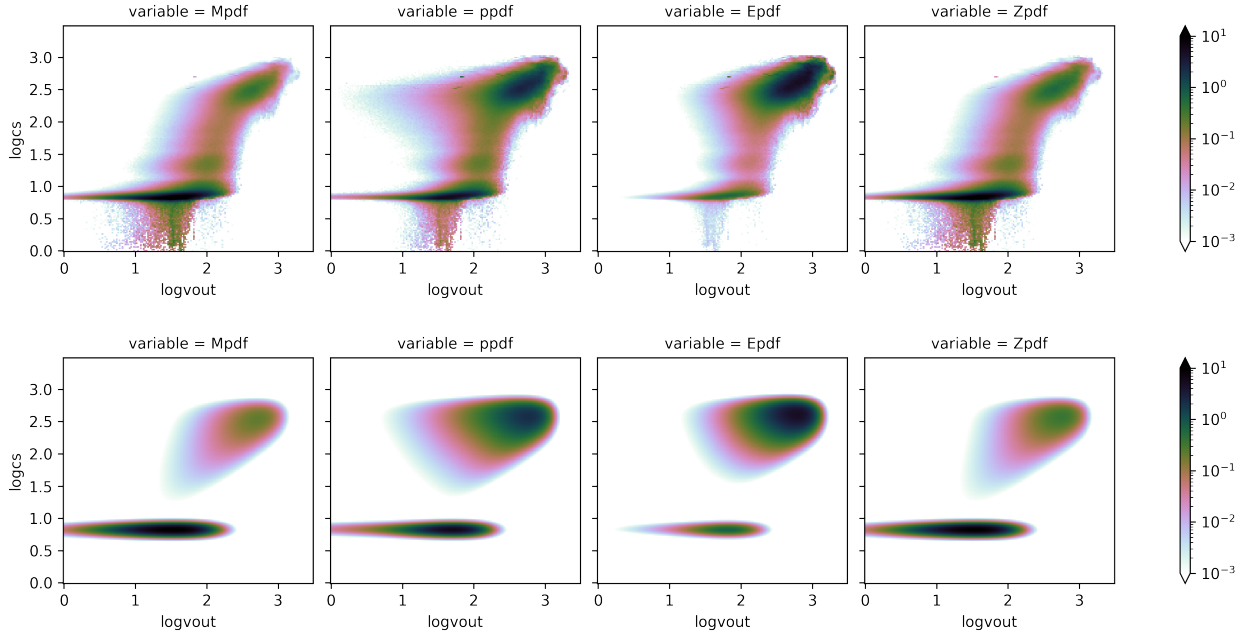
Since `TigressSimLoader` is a child class of `TigressWindModel`, this itself can be used to build model PDFs. If the simulation pdf is passed in `TigressWindModel.set_axes()` method, it will make axes identical to the simulation PDF.

```
sim.set_axes(pdf=sim.simpdf,verbose=True)
modelpdf=sim.build_model()
```

```
Setting up from simulation PDF...
  u in (0.0,3.5) with du = 0.02
  w in (0.0,3.5) with dw = 0.02
  Sigma_SFR = 0.0917, ZISM = 0.0302
  Mpdf : cool=0.867 int=0.046 hot=0.085 total=0.998
  ppdf : cool=0.381 int=0.057 hot=0.560 total=0.999
  Epdf : cool=0.044 int=0.018 hot=0.938 total=1.000
  Zpdf : cool=0.824 int=0.047 hot=0.127 total=0.998
```

```
# show all PDFs from R4 simulation
simpdf = sim.simpdf
simpdf[['Mpdf','ppdf','Epdf','Zpdf']].to_array().plot(col='variable',
    norm=LogNorm(vmin=1.e-3,vmax=10),
    cmap=plt.cm.cubehelix_r)
# this can be compared with model PDF
modelpdf[['Mpdf','ppdf','Epdf','Zpdf']].to_array().plot(col='variable',
    norm=LogNorm(vmin=1.e-3,vmax=10),
    cmap=plt.cm.cubehelix_r)
```

```
<xarray.plot.facetgrid.FacetGrid at 0x7fa83075e048>
```



Looking for more comprehensive comparisons? Check *Figures in Paper II*

None **Note:** This tutorial was generated from an IPython notebook that can be downloaded here.

1.8 Figures in Paper II

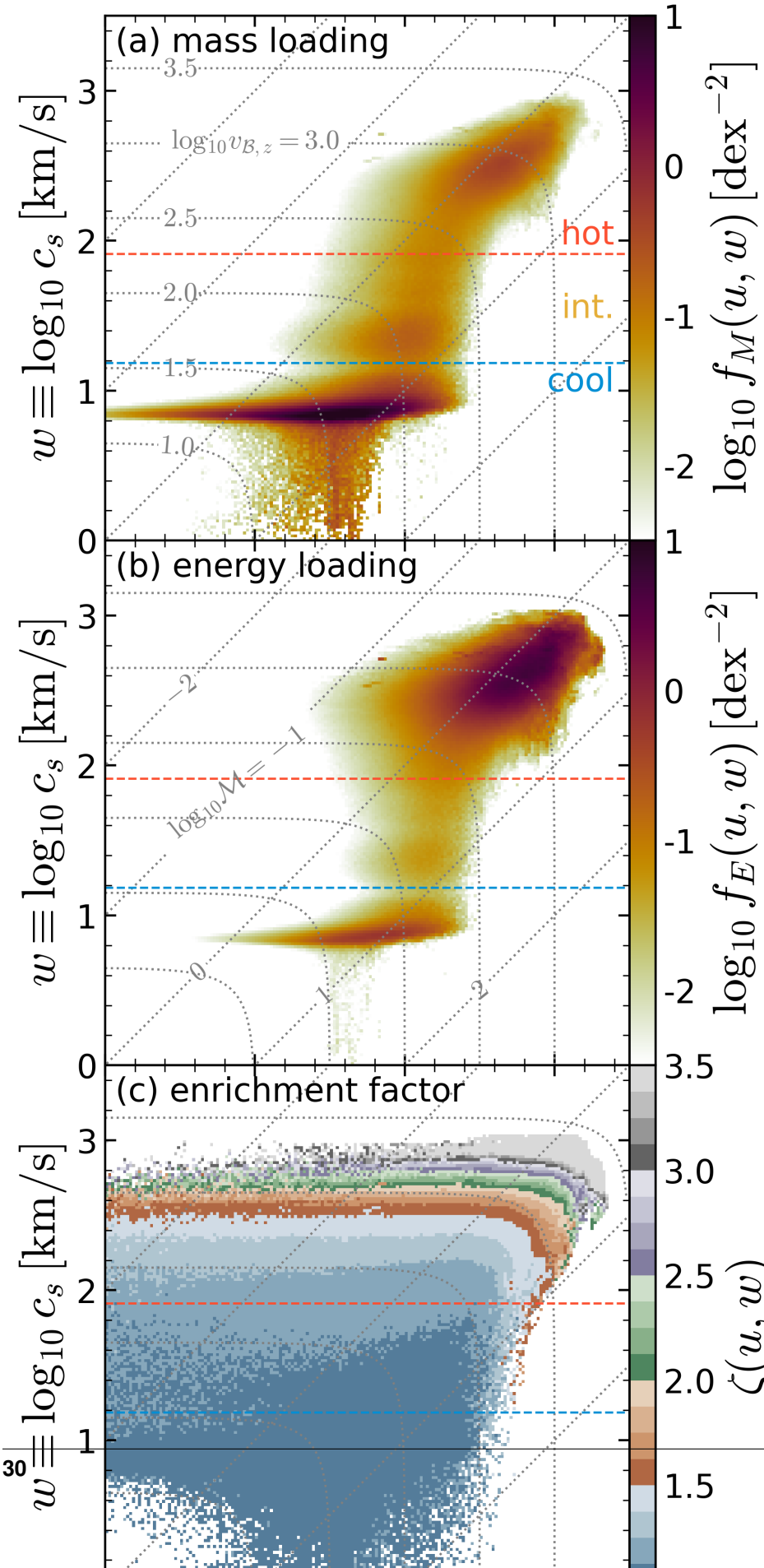
This tutorial shows how `Twind` package can be used to produce figures in Paper II. See the source code twind/plotter.py to know what is happening under the scean.

```
import twind
from twind.plotter import *
```

1.8.1 Figure 1: display simulation PDFs

```
# read in simulated PDF
sim = twind.TigressSimLoader('R4','H')
sim.load(download=True)
sim.set_axes(sim.simpdf)
```

```
with plt.style.context([{'axes.grid':False}]):
    fig=plot_flux_pdfs_yZ(sim.simpdf,grid=True)
```


(a) mass loading
3.5
$\log_{10} v_{B,z} = 3.0$
2.5
2.0
1.5
1.0
hot
int.
cool
(b) energy loading
$\log_{10}\mathcal{M} = -1$
(c) enrichment factor
$w \equiv \log_{10} c_s$ [km/s]
$\log_{10} f_M(u, w)$ [dex^{-2}]
$\log_{10} f_E(u, w)$ [dex^{-2}]
$\zeta(u, w)$

This figure shows the joint PDFs of $u \equiv \log_{10} v_{\rm out}$ and $w \equiv \log_{10} c_s$ for Model R4 at |z| = H. (a) Mass loading PDF, (b) energy loading PDF, and (c) enrichment factor. The red and blue dashed lines denote temperature cuts to separate cool ($T < 2 \times 10^4$ K), intermediate (2×10^4 K $< T < 5 \times 10^5$ K), and hot (5×10^5 K $< T$) phases. The dotted gray lines denote loci of constant Bernoulli velocity (labeled in (a))

$$v_{B,z} \equiv (v_{\rm out}^2 + 5c_s^2)^{1/2}$$

and Mach number (labeled in (b))

$$\mathcal{M} \equiv v_{\rm out}/c_s$$

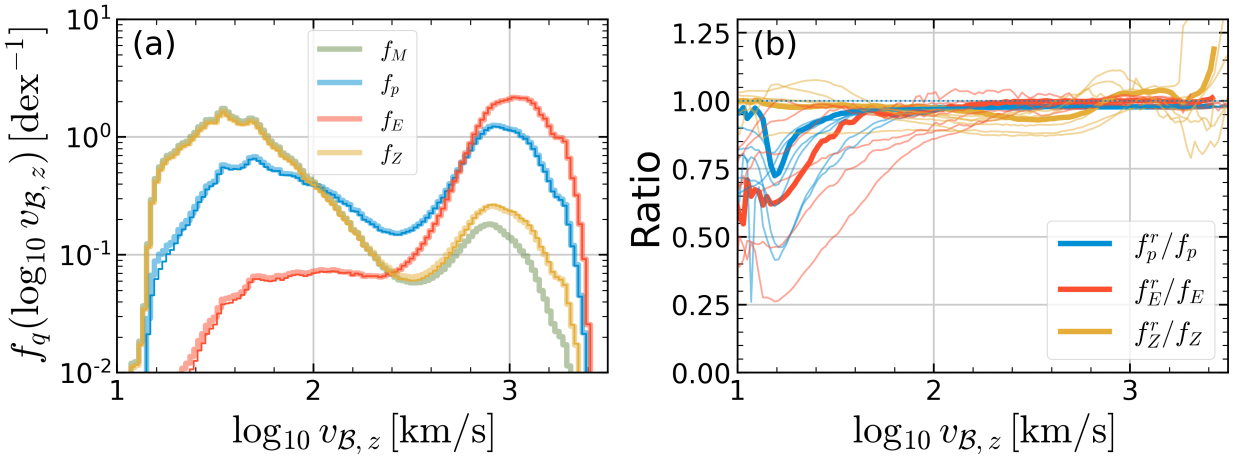
Notice that both mass and energy loading PDFs are distributed widely in large range of u and w, and there is clear correlation between the enrichment factor $\zeta(u, w)$ and Bernoulli velocity $v_{B,z}$.

1.8.2 Figure 2: reconstruct PDFs from mass loading PDF

As the joint PDF we are after is a function of the outflow velocity $v_{\rm out}$ and sound speed c_s, the momentum, energy, and metal loading PDFs can be reconstructed from the mass loading PDF. We present a reconstruction procedure in Section 3 of the paper. How good is this reconstruction procedure? See below.

```
sims=twind.TigressSimContainer(z0='H')
```

```
fig=flux_reconstruction(sims.sims)
```



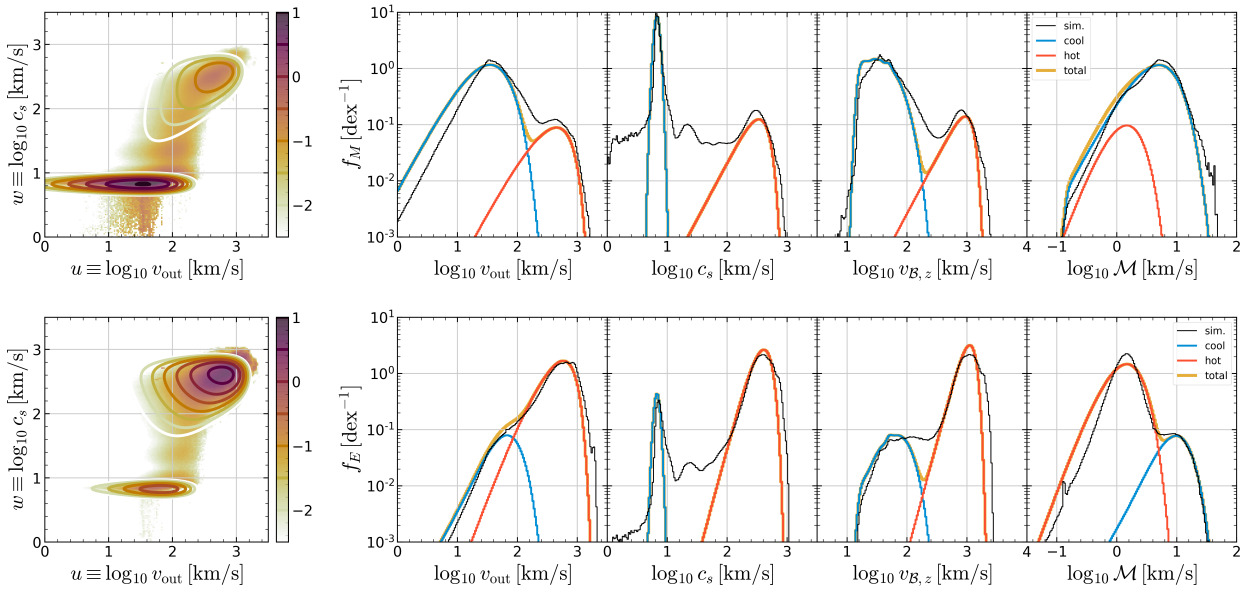
(a) Examples of PDFs for loading factors projected onto $\log v_{B,z}$ for model R4 at $|z| = H$. Thick lines show direct measurements of all PDFs, while thin lines with the same color (overlying the thick lines almost everywhere) show reconstructions from the mass PDF of momentum, energy, and metal PDFs.

(b) The ratios of reconstructed PDFs to the original PDFs for all models at $|z| = H$. The mean ratio at a given $\log v_{B,z}$ is obtained by a massflux weighted average. (Thick lines correspond to model R4, shown to left.

1.8.3 Figure 3: comparison between model and simulation PDFs

Joint PDF Model gives a mass loading PDF model for cool and hot outflows separately. The model PDF is entirely determined with two parameters: $\Sigma_{\rm SFR}$ and $Z_{\rm ISM}$. How good are the simple `Twind` model PDFs in comparison with the complex simulation PDFs? Here, we show multi-axes projection to give a quantitative view.

```
figM=comparison_pdfs(sims.sims['R4'],q='M')
figE=comparison_pdfs(sims.sims['R4'],q='E')
```



Comparison between simulated and model PDFs for R4: **(a)** mass loading and **(b)** energy loading. In each row, the first column shows full joint PDFs in logarithmic color scale ($\log f_{M,E}$ [dex^{-2}]) from the simulation (color) and model (contour). The remaining four panels are histograms showing projections onto (from left to right) **outflow velocity** $v_{\rm out}$, **sound speed** c_s, **Bernoulli velocity** $v_{B,z}$, **and Mach number** $\mathcal{M}$ axes. Model PDFs are separated into cool (blue) and hot (orange) components. The sum of the two (yellow) matches simulated PDFs (black lines) well (especially for dominating components).

1.8.4 Figure 4: Loading factor scalings

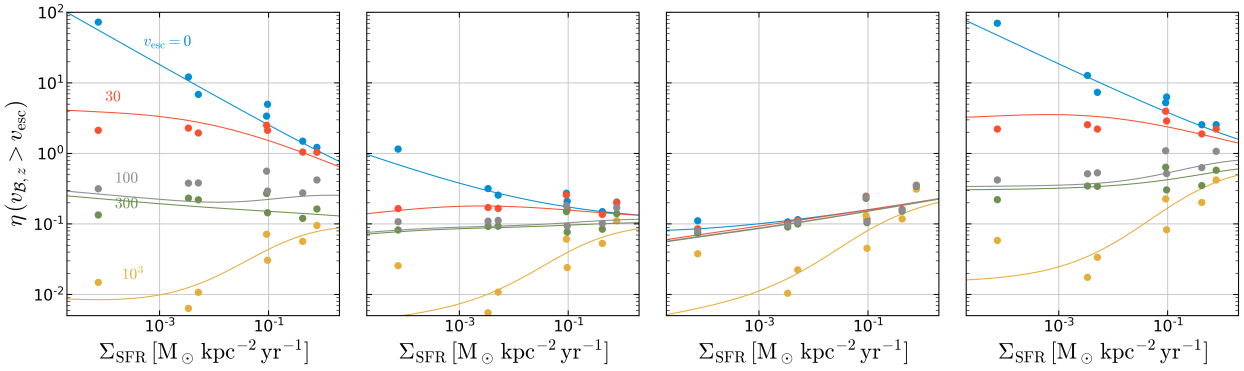
What is the model prediction for outflow properties as a function of $\Sigma_{\rm SFR}$? How much mass, momentum, energy, and metals can travel far from the launching position in a given galactic halo? Beyond the velocity-integrated loading factor scalings presented in Paper I, it is important to ask how much of outflows have specific energy $v_B^2/2$ large enough to climb up the potential well:

$$\eta_q(v_{\mathcal{B},z} > v_{\rm esc}) \equiv \tilde{\eta}_q \int_{v_{\mathcal{B},z}=v_{\rm esc}}^{\infty} \tilde{f}_q(u,w) du dw,$$

Depending on specific questions, one can use $v_{\rm esc} \equiv \sqrt{2\Delta\Phi}$ for gravitational potential difference between any distance, e.g., $\Delta\Phi = \Phi(R_{\rm vir}) - \Phi(H)$.

```
tw=twind.TigressWindModel(z0=sims.z0,verbose=False)
tw.set_axes(verbose=False)
modelpdf=tw.build_model(renormalize=True,energy_bias=True,verbose=False)
```

```
fig = show_loading(modelpdf,sims=sims.sims)
```



Loading factors for outflows with $v_{B,z} > v_{\rm esc}$. Filled circles are directly calculated from the simulation PDFs, while solid lines are from the model PDFs. Solid and dashed lines in (d) denote the model loading factors for $Z_{\rm ISM} = 0.02$ and 0, respectively.

Overall, the model tracks the general behavior of the simulation results.

Beyond the result at $|z| = H$ presented in Paper II, `Twind` includes model and simulation PDFs at $|z| = 2H$, 500 pc, and 1 kpc. Given its simplicity, the agreement with the simulation PDFs is stunning!

Figure 4 at z=2H

```
sims=twind.TigressSimContainer(z0='2H')
tw=twind.TigressWindModel(z0=sims.z0,verbose=False)
tw.set_axes(verbose=False)
modelpdf=tw.build_model(renormalize=True,energy_bias=True,verbose=False)
fig = show_loading(modelpdf,sims=sims.sims)
plt.setp(fig.axes,'ylim',(1.e-3,1.e2))
```

```
[0.001, 100.0, 0.001, 100.0, 0.001, 100.0, 0.001, 100.0]
```

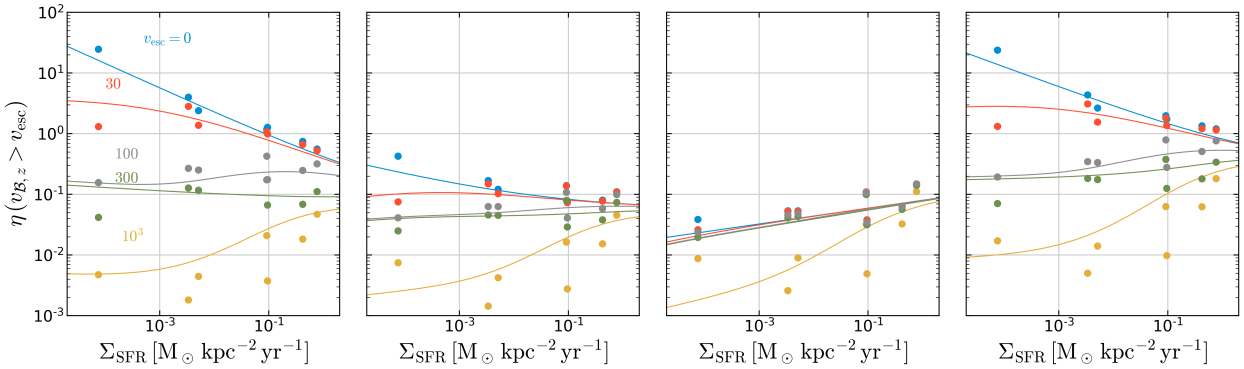


Figure 4 at z=500pc

```
sims=twind.TigressSimContainer(z0='500')
tw=twind.TigressWindModel(z0=sims.z0,verbose=False)
tw.set_axes(verbose=False)
modelpdf=tw.build_model(renormalize=True,energy_bias=True,verbose=False)
fig = show_loading(modelpdf,sims=sims.sims)
plt.setp(fig.axes,'ylim',(1.e-3,1.e2))
```

```
[0.001, 100.0, 0.001, 100.0, 0.001, 100.0, 0.001, 100.0]
```

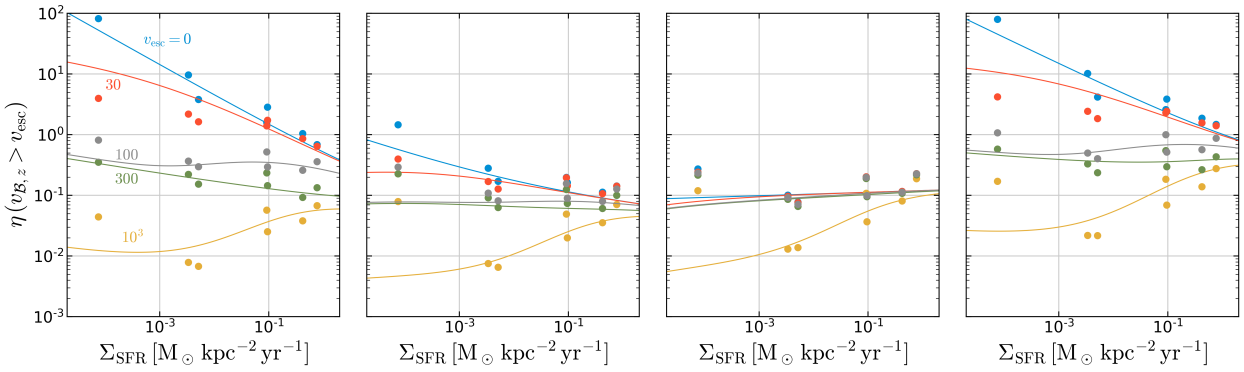
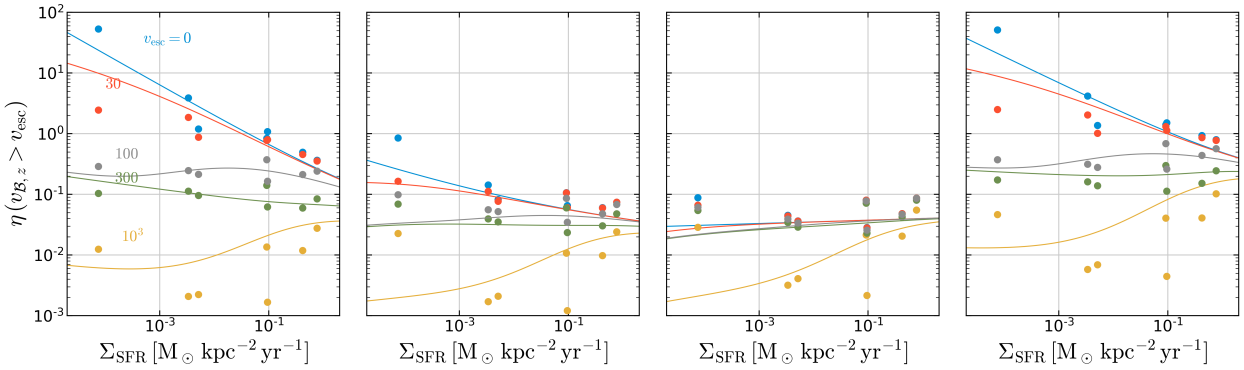


Figure 4 at z=1kpc

```
sims=twind.TigressSimContainer(z0='1000')
tw=twind.TigressWindModel(z0=sims.z0,verbose=False)
tw.set_axes(verbose=False)
modelpdf=tw.build_model(renormalize=True,energy_bias=True,verbose=False)
fig = show_loading(modelpdf,sims=sims.sims)
plt.setp(fig.axes,'ylim',(1.e-3,1.e2))
```

```
[0.001, 100.0, 0.001, 100.0, 0.001, 100.0, 0.001, 100.0]
```

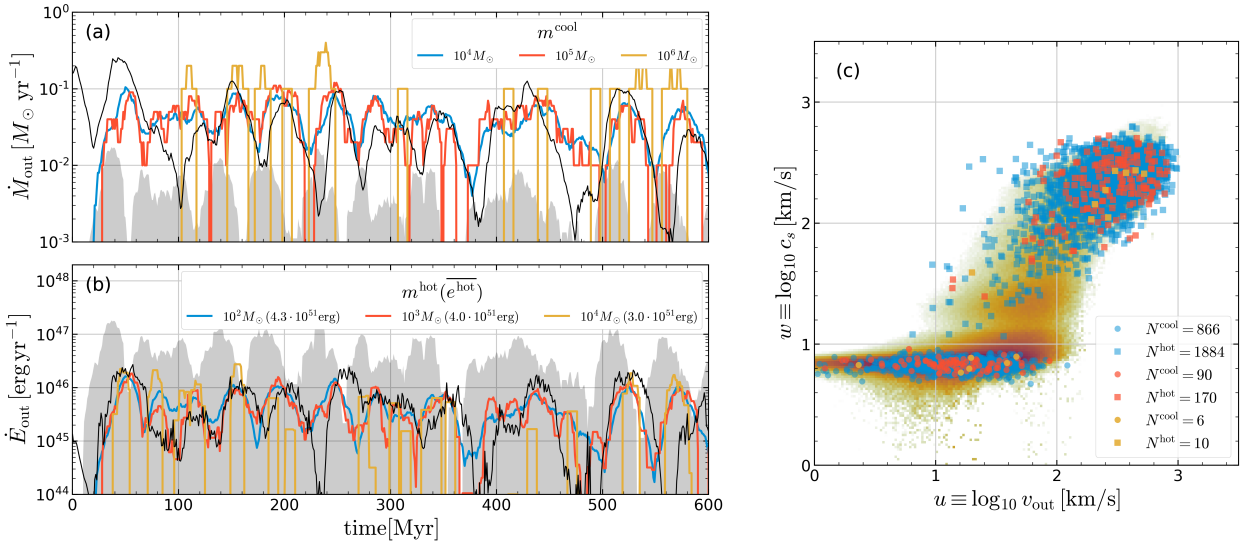


1.8.5 Figure 5: sampling example

Can we use `Twind` in cosmological simulations? Yes! We are working hard on developing subgrid model based on `Twind` in the SMAUG collaboration. Here's a quick demonstration of wind particle sampling. (See Appendix B of Paper II or the source code for the procedure in detail.)

```
# read in time series
sim = twind.TigressSimLoader('R8','H')
sim.load(download=True,time_series=True)
sim.set_axes(sim.simpdf)
```

```
fig = sampling_from_simulation_sfr(sim)
```



Model sampling demonstration for **(a)** mass outflow rate of cool gas and **(b)** energy outflow rate of hot gas. The simulation result (black solid) is compared to the model for three different particle mass choices (colored lines; see keys). The input to the model is $\Sigma_{\rm SFR}(t)$ from TIGRESS simulation R8, where SFR$= \Sigma_{\rm SFR} L_x L_y$ is shown as the grey shaded region in (a) and the corresponding SN energy injection rate is shown as the grey region in (b). **(c)** Distributions of cool (circles) and hot (squares) outflow particles sampled over t = 220 - 440Myr from the different mass sampling cases (number of particles drawn is shown in the legend). The simulation PDF over the same time interval is shown in the background.

1.9 API Reference

class twind.**TigressWindModel**(*z0='H'*, *verbose=False*)

TIGRESS Wind Launching Model class

Parameters **z0** (*['H','2H','500','1000']*) –

Examples

```
>>> from twind import *
>>> tw = TigressWindModel(z0='H')
>>> tw.set_axes()
>>> pdf = tw.build_Mpdf()
```

CoolMassFluxPDF(*u*, *w*, *sfr=1.0*, *params=None*)

Model of mass loading/flux PDF for cool gas

This utilizes generalized gamma (vout) and log-normal (cs) distributions.

Parameters

- **u** (*array_like (xarray.DataArray)*) – log vout
- **w** (*array_like (xarray.DataArray)*) – log cs
- **sfr** (*float, array_like*) – SFR surface density
- **params** (*array_like or None*) – (p_v, d_v, cs0, sigma) if None, *cool_params* attribute will be used

Returns **pdf**

Return type array_like (xarray.DataArray)

Notes

see *Joint PDF Model*

HotMassFluxPDF(*u*, *w*, *sfr=1.0*, *params=None*)
Model of mass loading/flux PDF for hot gas

This utilizes generalized gamma distributions in vBz and Mach, where vBz = sqrt(vout^2 + 5*cs^2) and Mach = vout/cs

Parameters

- **u** (*array_like (xarray.DataArray)*) – log vout
- **w** (*array_like (xarray.DataArray)*) – log cs
- **sfr** (*float, array_like*) – SFR surface density
- **params** (*array_like or None*) – (p_v, d_v, cs0, sigma) if None, *cool_params* attribute will be used

Returns **pdf**

Return type array_like (xarray.DataArray)

Notes

see *Joint PDF Model*

build_Mpdf(*verbose=False*)
Build mass loading/flux PDF

This will use axes attributes (*logvout*, *logcs*, *sfr*) set by *set_axes* method

Parameters **verbose** (*bool*) – print integrations of both cool and hot PDFs

Returns **pdfs**

Return type xarray.Dataset

build_model(*ZISM=None*, *renormalize=True*, *energy_bias=True*, *verbose=False*)
Build full PDFs for mass, momentum, energy, and metal PDFs

This will use axes attributes (*logvout*, *logcs*, *sfr*) set by *set_axes* method

Parameters

- **ZISM** (*float*) – set ZISM for metal PDF (if None, ZISM=0.02)
- **renormalize** (*bool*) – if True, momentum, energy, and metal PDFs are renormalized
- **energy_bias** (*bool*) – if True, apply energy bias factor in building the energy PDF
- **verbose** (*bool*) – print integrations of both cool and hot PDFs

Returns **pdfs**

Return type xarray.Dataset

reset_parameters(*z0*)
Reset parameters for different z0

This sets *z0* attribute and calls *_set_parameters()* method.

Parameters **z0** (*['H','2H','500','1000']*) –

set_axes(*pdf=None*, *sfr=(-6*, *2*, *100)*, *vout=(0*, *4*, *500)*, *cs=(0*, *4*, *500)*, *verbose=False*)
Set axes (*vout*, *cs*, *sfr*) using *xarray* for convenient manipulations

If a simulated pdf is an input, model axes are set to be identical to those of the input pdf otherwise, axes are set for a given (log min, log max, N bins)

Key attributes, *u* = *logvout*, *w* = *logcs*, *logsfr*, *vBz*, and *Mach*, will be set

Parameters

- **pdf** (*xarray.Dataset or xarray.DataArray*) – a joint pdf from simulation
- **sfr** (*float, tuple, list*) – a single value of SFR surface density, or log values of min, max, and No. of bins for SFR axis
- **vout** (*tuple, list*) – log values of min, max, and No. of bins for vout axis
- **cs** (*tuple, list*) – log values of min, max, and No. of bins for cs axis
- **verbose** (*bool*) – print input ranges

show_parameters()
Print all parameters in readable forms

class twind.**TigressWindSampler**(*z0='H'*, *verbose=False*)
Particle sampler for the TIGRESS Wind Model

Parameters **z0** (*['H','2H','500','1000']*) –

Examples

```
>>> from twind import *
>>> sampler = TigressWindSampler()
>>> cool,hot=sampler.draw_mass(sfr0,mcool,mhot,area=area,dt=dt)
```

draw_energy(*sfr*, *ecool*, *ehot*, *area=1.0*, *dt=1000.0*)
Draw particles with fixed particle energy quanta

Parameters

- **sfr** (*float, array_like*) – SFR surface density in Msun/yr/kpc^2
- **ecool** (*float*) – energy of cool gas in 10^51 erg
- **ehot** (*float*) – energy of hot gas in 10^51 erg
- **area** (*float*) – area in kpc^2
- **dt** (*float, array_like*) – time interval over which particle is sampled

Returns **cool, hot** – dicts containg particle mass, 3 component velocity, sound speed, metallicity, and index of each particle in the corresponding input SFR surface density array, which will be used for reconstuction of time series

Return type dicts

draw_mass(*sfr*, *mcool*, *mhot*, *area=1.0*, *dt=1000.0*)
Draw particles with fixed particle mass quanta

Parameters

- **sfr** (*float, array_like*) – SFR surface density in Msun/yr/kpc^2
- **mcool** (*float*) – Mass of cool gas in Msun
- **mhot** (*float*) – Mass of hot gas in Msun
- **area** (*float*) – area in kpc^2
- **dt** (*float, array_like*) – time interval in yr over which particle is sampled

Returns **cool, hot** – dicts containg particle mass, 3 component velocity, sound speed, metallicity, and index of each particle in the corresponding input SFR surface density array, which will be used for reconstuction of time series

Return type dicts

get_refs(*sfr*)

Obtain reference rates and loading factors for a given SFR surface density using scaling relations

Parameters **sfr** (*array_like*) – SFR surface density

Returns

- **refs** (*array_like*) – reference mass, momemtum, energy, metal outflow rates
- **eta** (*array_like*) – mass, momemtum, energy, metal loading factors for total gas
- **eta_cool** (*array_like*) – mass, momemtum, energy, metal loading factors for cool gas
- **eta_hot** (*array_like*) – mass, momemtum, energy, metal loading factors for hot gas

twind.**to_time_series**(*p*, *time*)

Function to convert the particle data into time series

Parameters

- **p** (*dict*) – paticle data as returned by *TigressWindSampler.draw* method
- **time** (*array_like*) – time array corresponding to SFR time series used to sample particles

Returns **out** – time series of mass, momemtum, energy, and metals carried by sampled particles

Return type (m, p, E, mZ)

class twind.**TigressSimContainer**(*z0='H', modelnames=['R2', 'R4', 'R8', 'R16', 'LGR2', 'LGR4', 'LGR8']*)

Simulation PDF container for the TIGRESS simulation suite

Load all models at a given height.

Parameters

- **z0** (*['H','2H','500','1000']*) –
- **modelnames** (*['R2','R4','R8','R16','LGR2','LGR4','LGR8']*) – list of model names to load

Examples

```
>>> from twind import *
>>> sim = TigressSimContainer(z0='H')
```

class twind.**TigressSimLoader**(*name='R4'*, *z0='H'*)
Simulated PDF loader for the TIGRESS simulation suite

Parameters

- **name** (*['R2','R4','R8','R16','LGR2','LGR4','LGR8']*) –
- **z0** (*['H','2H','500','1000']*) –

Examples

```
>>> from twind import *
>>> sim = TigressSimContainer(model='R4',z0='H')
```

download(*source='tigressdata'*, *time_series=False*)
Download simulation pdf data

Parameters **source** (*['tigressdata','dataverse','cca']*) –

Note: 'cca' server is not yet available as a source

load(*download=False*, *time_series=False*)
Load simulation PDF

Parameters **download** (*bool*) – automatically call download() method if file doesn't exist

pdf_reconstruction()
PDF reconstruction from mass loading PDF

CHAPTER 2

License & Attribution

Twind is free software made available under the MIT License.

If you make use of **Twind** in your work, please cite our papers:

- `Kim et al. 2020a, ApJ, 900, 61` *First results from SMAUG: Characterization of Multiphase Galactic Outflows from a Suite of Local Star-Forming Galactic Disk Simulations* [arXiv, ADS, BibTeX]
- `Kim et al. 2020b, ApJL submitted` *TIGRESS Multiphase Wind Launching Model* [links].

Python Module Index

t

Index