Effect of Grain Growth on Dust content of photoevaporative flows in protostellar discs

George Dadunashvili

SIGILL·UNIVERSITAT·LUDOVIC·MAXIM·

Effect of Grain Growth on Dust content of photoevaporative flows in protostelar discs

Effekt des Staubteilchen-Wachstums, auf den Staubdgehalt des photoevaporativen Flusses der protostelaren Scheiben

George Dadunashvili

Bachelorarbeit
an der Fakultät für Physik
der Ludwig-Maximilians-Universität
München

Betreuer:
Prof. Dr. Barbara Ercolano
und
MAoSc. Giovanni Rosotti

August 1, 2017

Contents

1. Introduction

This thesis is about photoevaporation in edge on discs, driven by Extreme Ultraviolet Radiation (EUV). It mainly refers to the simulations carried out by James E. Owen (Owen et al., 2011). The main motivation for this work is to understand why Owen's simulations partially failed to reproduce the observations made on the object PDS 144N (Perrin et al., 2006) and to explore possibilities to correct the outcome, by altering the initial conditions.

Owen first calculated the photoevaporative mass flow, leaving the disc, which delivered data about maximum size of grains entrained in the wind, for each radius. Final step in his hydrodynamic simulations was to calculate the dust density distribution above the mid plane of the disc. The later one was used to compute synthetic disc images with MocassinThinimmage (an altered version of radiative transfer code Mocassin). Calculated images correctly reproduced the morphology of the disc ('wingnut' morphology), however the color of the images was dominated by blue light (see Owen et al. 2010 Figure 4), which is in conflict with observations, which show that at larger radii and height above the mid plain, preferably red light is scattered.

One plausible explanation of this deviation between synthetic images and observations lies in the grain size distribution inside the disc, which was assumed to be distributed according to the MRN power law.

$$\frac{dN}{da} \propto a^{-3,5}$$

One can see that this distribution is dominated by small grains, so for each large grain there is a great number of small grains also entrained in the wind. The result of this behaviour is a dust density distribution where the role of larger grains is almost negligible. As we know that red light is scattered by larger grains, the outcomes of the simulations are not surprising any more.

In order to test if grain size really is the decisive quantity for the end result of the calculations, we need to take a new distribution, which is a result of grain growth simulations, calculate the dust density, and compute new synthetic images. In this work only the first step will be discussed. Even if the calculation of synthetic images is necessary for direct comparison with observational data, a new dust density distribution will already give a hint, to test the previously introduced heuristic explanation.

Before one changes the initial conditions of Owen's simulations, his previous results need to be reproduced. The code used for this calculations will be introduced in the appendix, the results of the calculations will be discussed and compared to the outcomes of Owens simulations in following chapters.

2. Theoretical Model

In this chapter, the main theoretical methods, used for calculations will be displayed. In two subsections the reader first will be introduced to the force equations, governing the dynamics of the mass flow, and then to the set of input data, used for simulations in this model. Here is also to mention that, in this thesis, as in Owen's paper only the extreme ultraviolet (EUV) photoevaporative flow is considered. A very pleasing side effect of the EUV regime is that the process is isothermal, so one does not have to care about the temperature distribution, as it is the same everywhere. This fact makes simulations considerably easier, in comparison with other regimes (e.g. X-ray regime).

If more information, about the process of photoevaporation is desired by the reader, the following Paper (Armitage, 2011) gives an informative and well understandable overview of the topic.

2.1. Dynamics

In this section the forces involved in the process will be discussed and the main equation used in the source code to calculate the grain size will be introduced.

In our case the equation of forces acting on a single grain entrained in the wind is the following one:

$$\overrightarrow{F}_{tot} = \overrightarrow{F}_{G} + \overrightarrow{F}_{d} + \overrightarrow{F}_{rot} \tag{2.1}$$

where the $\overrightarrow{F}_{G}$, $\overrightarrow{F}_{d}$ and $\overrightarrow{F}_{rot}$ are, respectively: gravitational force caused by the star, the drag force caused by the photoevaporative wind and centrifugal force caused by the rotation of the protostellar disc. The equations of gravitational and centrifugal forces are common knowledge and do not need to be displayed here separately. However for the drag force the same equation from Owen's paper will be adopted. For the origin of the equation see (Takeuchi et al., 2005). The drag force is given by:

$$F_d \approx \frac{m_d \rho_w v_w^2}{\rho_d a} \tag{2.2}$$

where m_d , ρ_d and a are, respectively: mass, density and radius of a single dust grain, which we assume to have spherical form. ρ_w and v_w are the density and the velocity of the photoevaporative wind.
After writing down all forces, the grain mass m_d cancels out and we obtain the following Equation.:

$$G\frac{M_*}{r^2} = \frac{\rho_w v_w^2}{\rho_d a} + \frac{v_{rot}^2}{r} \tag{2.3}$$

where M_* is the mass of the star, v_{rot} the rotational velocity of the disc and r the radial distance from the center of the disc to the grain.
The primary task now is to calculate which grains can be entrained in the wind. In order to do so one rewrites the equation (2.3) for the grain radius a. We denote $a(r)_G \equiv G\frac{M_*}{r^2}$ for gravitational acceleration and $a(r)_{rot} \equiv \frac{v_{rot}^2}{r}$ for rotational acceleration and get:

$$\begin{aligned} a_G &= \frac{\rho_w v_w^2}{\rho_d a} + a_{rot} \\ \frac{\rho_w v_w^2}{\rho_d a} &= a_G - a_{rot} \\ a &= \frac{\rho_w v_w^2}{\rho_d (a_G - a_{rot})} \end{aligned}$$

As this problem has cylindrical symmetry, one only needs two components of the wind velocity; v_r and v_θ. So we finally obtain an equation for $a(r)_{max}$, a maximum grain size which can still be entrained in the wind (or streamline) at the given radius from the star.

$$a(r)_{max} = \frac{\rho_w (v_r^2 + v_\theta^2)}{\rho_d (a(r)_G - a(r)_{rot})} \tag{2.4}$$

Note that this equation is only valid for a given streamline at the starting point of the streamline. To understand this point one has to go back to the equation 2.3. One can assume that the wind flow is approximately spherical at large radii ($z/R > 1$ where R is spherical radius) (see Figure 2.1), this implies that $\rho_w v_w$ falls off as $1/r^2$. Wind velocity also can be regarded as monotonically growing for larger radii (see Figure 2.2). So the drag and centrifugal forces both fall off as $1/r$ and always dominate over the gravitational force (falling off as $1/r^2$), at large radius. This means that if a grain is once entrained in the wind it will stay entrained and be carried out at large distances. But this also means that the domination of the drag force over the gravitational force is getting stronger, so at larger radii grains bigger then a_{max} could be entrained in the wind. This does not happen because there are no larger grains then a_{max}, as they could not be entrained at the beginning of the streamlines, and carried out to far distances.

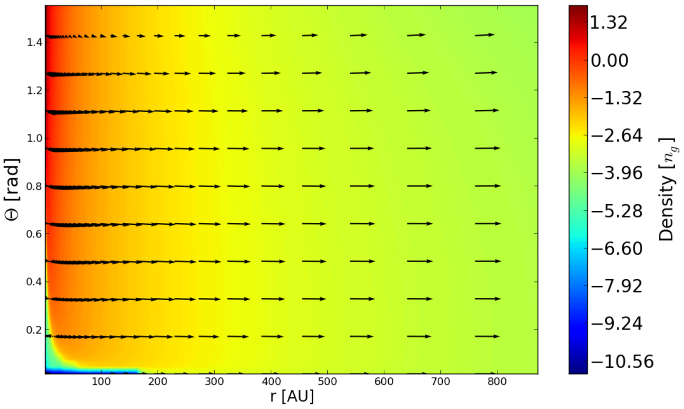

Figure 2.1.: gas density distribution (spherical coordinates): the color map shows the density distribution of the gas, while the arrows show the magnitude and direction of the wind velocity. One can clearly see that at large radii gas flow is spherical.

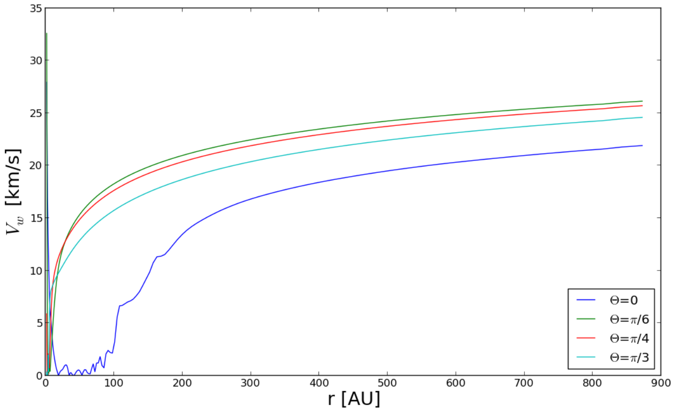

Figure 2.2.: magnitude of the velocity of the wind(for different Θ): one can see that the magnitude of the velocity has a approximately linear dependence on the cylindrical radius (for large r).

The equation makes possible determine the dust content of the gas flow. Only one further step is needed to determine the dust density (the final goal of this calculations).

Dust density, for MRN distributed grain sizes

If one assumes that that the grain size is distributed according to the MRN power law, the dust density can be calculated analytically.

The number density of grains with radius a is given by:

$$\frac{dn}{da} = ca^{-3.5} \quad \text{with c as proportionality constant} \tag{2.5}$$

The mass for a spherical grain is: $m_d(a) = \frac{4}{3}\pi\rho_d a^3$, so the differential mass is: $dm = 4\pi\rho_d a^2 da$. In order to calculate the total density, one has to integrate over the number density, for each mass bin:

$$\rho_{dust} = \int_{m_{min}}^{m_{max}} \frac{dn}{da} dm \tag{2.6}$$

$$\rho_{dust} = \int_{m_{min}}^{m_{max}} \overbrace{\frac{dn}{da}}^{ca^{-3.5}} \underbrace{dm}_{4\pi\rho_d a^2 da}$$

$$= 4\pi\rho_d c \int_{a_{min}}^{a_{max}} a^{-1.5} da$$

$$\rho_{dust} = 8\pi\rho_d c \; [a_{min}^{-0.5} - a_{max}^{-0.5}]$$

,

Now one can use the relation for dust to gas ratio, to determine the the constant c:

$$\rho_{dust} = \epsilon\rho_{gas} \quad \text{in our case } \epsilon = \frac{1}{100} \tag{2.7}$$

In this equation now the expression for ρ_{dust} can be inserted:

$$\rho_{dust} = 8\pi\rho_d c\ [a_{min}^{-0.5} - a_{max}^{-0.5}] = \epsilon\rho_{gas}$$

$$c = \frac{\epsilon\rho_{gas}}{8\pi\rho_d\ [a_{min}^{-0.5} - a_{max}^{-0.5}]}$$

Now the proportionality constant c is determined, and the dust density distribution can be calculated, however one has to take in account that not all grains can be entrained in the wind, so the maximum grain size depends on the cylindrical radius r. The way to calculate $a(r)_{max}$ will be introduced in he chapter 3.1 (see Figure 3.2).

$$\begin{aligned}\rho(r)_{dust} &= 4\pi\rho_d \overbrace{\frac{\epsilon\rho_{gas}}{8\pi\rho_d\ [a_{min}^{-0.5} - a_{max}^{-0.5}]}}^{c} \int_{a_{min}}^{a(r)_{max}} a^{-1.5} da \\ &= \frac{\epsilon\rho_{gas}}{2\ [a_{min}^{-0.5} - a_{max}^{-0.5}]} 2\ [a_{min}^{-0.5} - a(r)_{max}^{-0.5}]\end{aligned}$$

One finally receives the equation for the dust density distribution in the gas flow:

$$\rho(r)_{dust} = \frac{\epsilon\rho_{gas}}{[a_{min}^{-0.5} - a_{max}^{-0.5}]}\ [a_{min}^{-0.5} - a(r)_{max}^{-0.5}] \tag{2.8}$$

This is the equation used in the code (see appendix B; code line 51) to calculate the dust density images (see Figure 3.4).

Dust density distribution, after grain growth

New grain size distribution is adopted from (Birnstiel et al., 2011) and (Birnstiel et al., 2012) grain growth simulations Simulations. Due to a lucky accident the grid of radial coordinates r is very similar for both Owen's and (tills) simulations (Average relative difference between two different r-axis points is 1.44%), so no interpolation between new and old coordinate grids was needed.

Birnstiel's simulations provide one with dust density for every given grain size at every given point in the protostellar disc. As it is hard to say where exactly in the disc the photoevaporative wind is launched, one has to define the launching surface arbitrarily. This fact does not constitute a problem, as one can see that with variation of the launching surface the outcomes of simulations do not vary much. In this case it is much easier to calculate the dust density distribution, as the maximum grain size is already known (it is the same for both cases), and the data provide the dust densitiy for each grain size. To determine the total density, one needs to sum up the densities for individual grain sizes:

$$\rho_{dust,tot} = \sum_i \rho_{dust,i} \quad \text{for } \underline{\text{all}} \ a_i \tag{2.9}$$

Now one has to take in account that not all grains can be entrained in the wind:

$$\rho(r)_{dust,cutoff} = \sum_i^{a(r)_{max}} \rho(r)_{dust,i} \quad \text{for each } a(r)_i < a(r)_{max} \tag{2.10}$$

The ratio of this two densities is the relevant quantity in order to calculate the density distribution of the dust content entrained in the wind $\rho(r)_{dust}$. Just like in the MRN case we obtain the equation:

$$\rho(r)_{dust} = \epsilon \frac{\rho(r)_{dust,cutoff}}{\rho_{dust,tot}} \rho_{gas} \quad \text{with } \epsilon = \frac{1}{100} \tag{2.11}$$

2.2. Simulation Parameters

In the simulations conducted for this thesis, all parameter were adopted from (Owen et al., 2011) and (Font et al., 2004).

The system consists of a young $2.5 M_\odot$ star and its disc, and is determined by the parameters listed in the table (2.1). Also the input data for the simulations, like the gas density distribution and coordinate grid were the same as in Owen's simulations(a spherical grid with $\Theta = [0; \pi/2]$ and $r = [0; 40] r_g$ with logarithmically spaced radial coordinates, to provide high resolution in the region $r \sim r_g$, where the mass loss rate has its maximum). For more details see the source code (Appendix 1), where all the relevant constants and variables are commented, this should make the structure of the code easy to understand.

It is crucial to understand the difference between a_{max} and $a(r)_{max}$. The Former one is the maximum grain size, available in the disc, same at every distance. Its value in MRN case is $1mm$ and in the case after grain growth $1cm$. The latter is the maximum grain size which can be entrained in the wind, this value is simply a solution of the equation 2.4. It remains same for both simulations.

Parameter		Value [units]
mass of the star	M_*	$2.5M_\odot$
ionizing luminosity	Φ	10^{43} $[s^{-1}]$
speed of sound in ionized gas	c_s	10^6 $[cm/s]$
length scale	r_g	$\frac{GM_*}{c_s^2} = 22.18$ $[AU]$
scale parameter for number density	n_g	$0.1\left(\frac{3\Phi}{4\pi\alpha_2 r_g^3}\right)$
mean mass of ionized hydrogen atoms	$\bar{m}_H$	$1.35m_H$
scale parameter for gas density density	ρ_g	$\bar{m}_H n_g$
density of a single dust grain	ρ_d	1 $[g/cm^3]$
maximum grain size available in the disc (MRN case)	a_{max}	1 $[mm]$
minimum grain size available in the disc (MRN case)	a_{min}	5×10^{-3} $[\mu m]$

Table 2.1.: Simulation parameters

3. Implementation of the Model

In chapter 2 and its subsections, all information needed for understanding the theory and the way of its implementation has already bin provided, so in the following sections (3.1 and 3.2) the results of the simulations can be discussed and interpreted.

3.1. Benchmarking

In this section the focus is laid on reproducing the results of (Owen et al., 2011) As described in Chapter 2 the first step is to calculate the maximum grain size entrained in the wind. To do so one needs to calculate the mass flow, which effectively are the streamlines (Figure 3.1) (or integral lines) of the velocity vector field (Figure 2.1) (for source code see appendix A subsection **calculation of streamlines**).

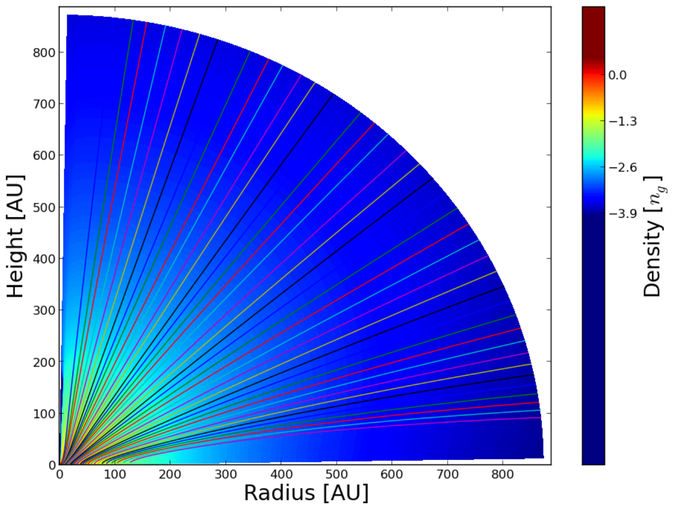

Figure 3.1.: Streamlines: the color map shows the gas density, while the streamlines display the mass flow (the color of the streamlines is arbitrary).

All simulations were conducted with 40 streamlines. Once the streamlines are known, the maximum grain size ($a(r)_{max}$) for each one of them can be easily calculated (see Appendix A; source code lines from 181 to 245). So one obtains maximum grain size as a function of the cylindrical radius (see Figure 3.2)

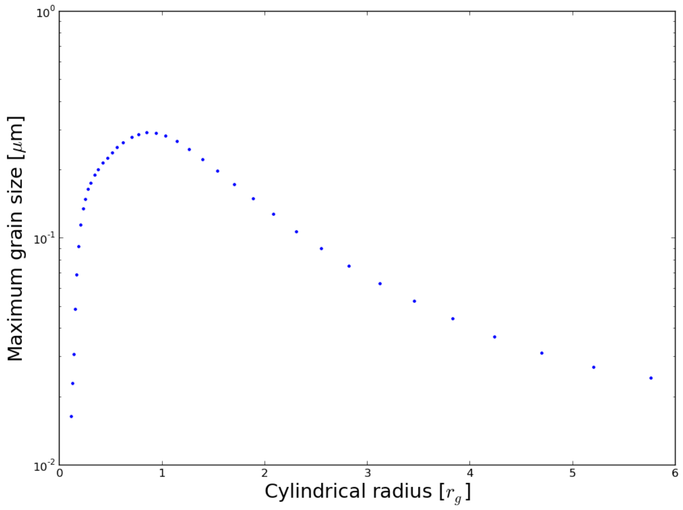

Figure 3.2.: Maximum grain size, at each starting point of a streamline. Note that the function has its maximum around $r \sim r_g$. This behaviour was expected, as in this region the mass loss rate for EUV driven photoevaporative wind is maximal.

This value remains the same all along the streamline and can be interpolated between them (see Figure 3.3). This provides one with the necessary values of a_{max} at each point in the wind. So the dust density distribution can be calculated. Unfortunately at this point the results of the simulation deviate from the results presented by Owen et al. by one order of magnitude. As the original source code used in Owen's simulations is not available any more, it is impossible to compare the two simulations directly and determine, if the deviation is caused by a single error or by the difference between the two methods. At this points the reader can only find comfort in the fact that the source code, used for the purposes of this thesis, was tested several times and no errors were found. However one can never prove the absence of a mistake, so every reader is invited to recheck the code by him-/herself. Luckily this deviation does not constitute a big problem after all. As the same values of a_{max} are used in both simulations (MRN case and case after grain growth), one has same systematic error in both results, so it still makes sense to compare them. Here should also be noted that, beside the deviation discussed above the morphology of all distributions match perfectly with the results of (Owen et al., 2011).

As discussed in the chapter 2.1 the distribution of maximum grain size in the

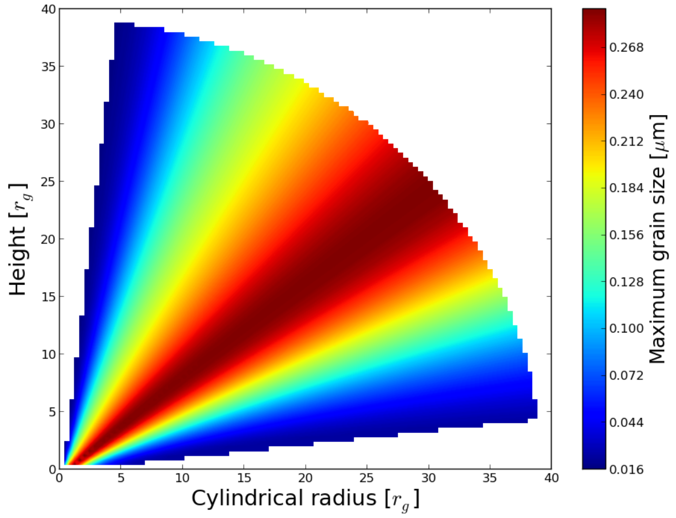

Figure 3.3.: maximum grain size distribution in the wind.

wind, is all one need to calculate the dust density distribution, Figure 3.4 (see Appendix B).

3.2. Variation of grain size distribution in the disc

After all plots presented in the paper of (Owen et al., 2011) are reconstructed and the results are satisfying, one can move on to the actual task of this thesis. Testing the effect of grain growth on the dust density distribution in the photoevaporative wind. The steps are the very similar as in the first part:

1. define the launching surface in the disc
2. calculate the maximum grain size $a(r)_{max}$ at the starting points of all streamlines
3. interpolate the $a(r)_{max}$ values between the streamlines
4. use the formula for the dust density (Equation 2.11) to calculate the distribution
    a) calculate the ratio of total and cut-off dust densities, on the launching surface

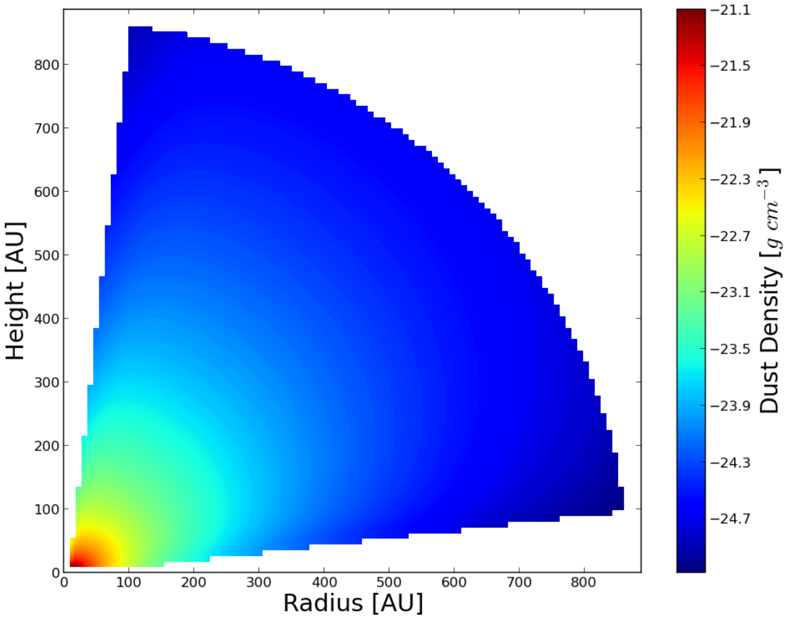

Figure 3.4.: dust density distribution(MRN case).

b) interpolate the values of density ratio between the streamlines

c) insert the results in the formula for the dust density (Equation 2.11)

The source code used for the MRN case is identical in steps 2. and 3. and can be used here without any variation. The interesting part begins with the calculation of dust density ratio (see appendix A, source code lines; from 306 to 343). The result is displayed in Figure 3.5.

Now all preparations are done to perform the final step, use the Equation2.11 and calculate the new dust density distribution which is displayed in Figure 3.6.

Note that the maximum value of the density in the case after grain growth is lower then in MRN case. This is exactly what one would expect, as one has less smaller grains, which were responsible for high densities in MRN case. I order to see the difference between the two cases more clearly, one can look at Figure 3.7. Here each point stands for a grain group of same size, one can see which group contributes to which density. In the MRN case (blue dots) one can clearly see that each density region contains all kinds of grains, so for each large grain there are more smaller grains and for all densities the role of small grains dominates over the role of larger ones. Whereas after grain growth (green dots), large grains are solely responsible

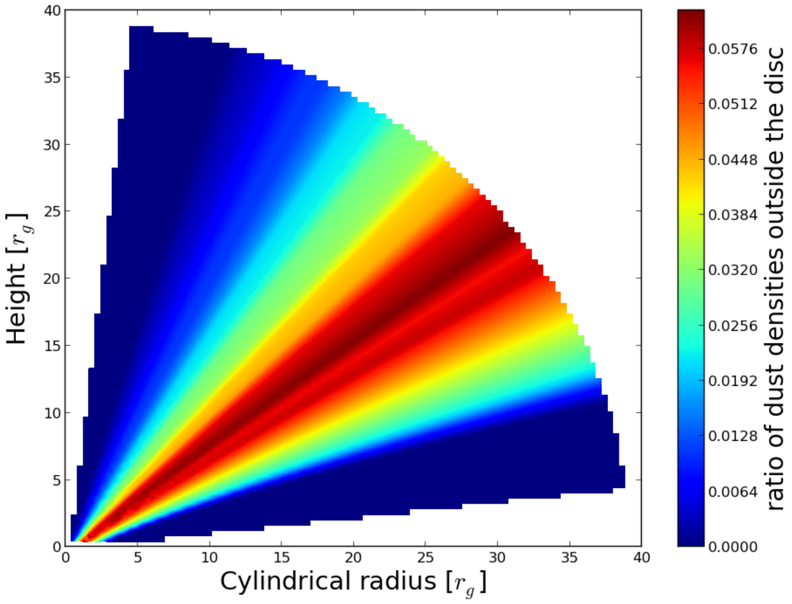

Figure 3.5.: ratio of dust densities.

for high density regions, and small grains have less importance. This results deliver reasons to belive that grain growth can indeed explain deviations between synthetic images calculated by (Owen et al., 2011) and the observed object PDS 144N.

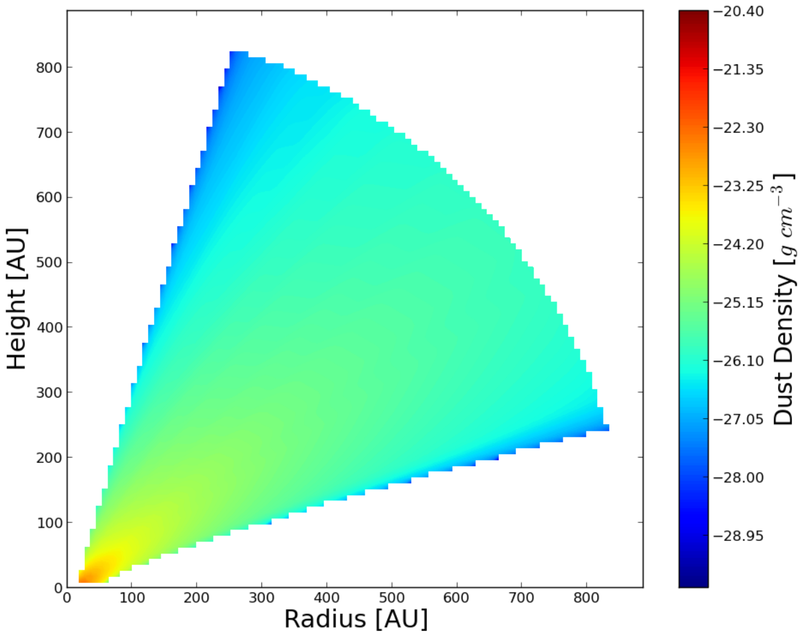

Figure 3.6.: dust density distribution(after grain growth).

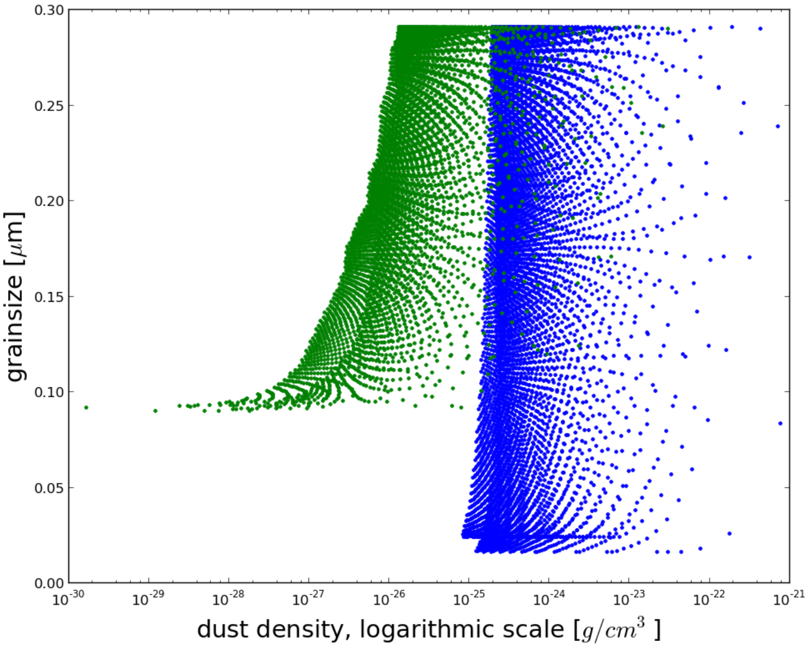

Figure 3.7.: grain size plotted against corresponding dust density bin. Blue pints are for MRN case and green pints for the 'grain growth' case.

4. Conclusion and Outlook

In this thesis, the reader was introduced to hydrodynamic simulations, of EUV driven photoevaporative wind in protostellar discs. The main goal of this simulations were to determine which effect grain growth has on the dust content of photoevaporative gas flow. Here first the results of Owen et al. (2010) were reproduced and then compared with new simulation results where the grain size distribution in the disc were altered. This was not done arbitrarily, just in order to save the outcomes of (Owen et al., 2011), but is very well justified physically, as grain growth in the disc is expected and is one of the early steps for the formation of planets. // As final conclusion one can say that, the plausibility of the initial assumption, that grain growth will resolve the problem (deviation between observations and synthetic images calculated by Owen et al. 2010), were confirmed. However note that, between plausibility and correctness still is a large gap. In order prove that, the new dust density distribution is correct, one need to calculate the synthetic images for the simulations with bigger grains and compare them with observations.

Appendices

A. Source Code for calculation of maximum grain size

All source codes displayed in Appendix A are derived from the code for simulation of X-ray driven photoevaporation in low mas star systems, provided by Giovanni Rosotti. The main part of the code was altered for purposes of this thesis and made suitable for simulations in EUV regime. However, codes for bilinear interpolation and streamline calculation were used directly, as they are independent of the photoevaporation regime and do not need any alternation. All parts of the code are written in the programming language PYTHON2.7.

```
from __future__ import division
import pyximport; pyximport.install()
import bilinear
import matplotlib.pyplot as plt
import matplotlib.mlab as mlb
import numpy as np
import streamlines



#constants (in CGS)
#---------------------------------------------------------
m_p       = 1.6726231e-24        # proton mass in g
mu_bar    = 1.37125
gamma     = 5./3.
k_b       = 1.380658e-16         # Boltzmann constant in erg/K
rho_dust  = 1.                   # g cm^-3
Grav      = 6.67259e-8           # gravitational constant in cm^3 g^-1 s^-2
M_sun     = 1.989e33             # mass of the sun in g
M_star    = 2.5*M_sun
AU        = 1.496e13             # astronomical unit in cm

u         = 1.660538782e-24      # atomic mass unit in g
c_s       = 1.0e6                # Speed of sound in ionized gas

#the imported Data is in scaled units
#Scale parameters (in CGS):

r_g    = ( Grav * M_star ) / (c_s**2)   # length scale
print 'r_g [AU]'
print r_g
print r_g/AU
alpha2 = 2.6e-13                        # recombination coeff. for all states except ground state
m_H    = 1.008 * u                      # Hydrogen mass
m_mean = 1.35  * m_H                    # mean mass per hydrogen atom
PHI    = 1e43                           # ionizing flux s^-1

    #particle density scale
n_g    = 0.1*((3*PHI) / (4*np.pi*alpha2*(r_g**3)))**(1/2)

print 'n_g'
print n_g
ng     = 2.8*(1e4)*((PHI/(1e41))**(0.5))*((M_star/M_sun)**(-1.5))
print 'ng'
print ng
print 'n_g - ng :' , n_g - ng

rho_g  = m_mean * n_g
print 'rho_g'
print rho_g
#-----------------------------------------------------------------

#Import Data

r     = np.loadtxt("inpdata/radius.dat") * r_g
th    = np.loadtxt("inpdata/theta.dat" )
rho   = np.loadtxt("inpdata/density.dat"     ,delimiter = ",") * rho_g
vr    = np.loadtxt("inpdata/velocity_r.dat"  ,delimiter = ",") * c_s
print 'vr: ' , vr
vth   = np.loadtxt("inpdata/velocity_th.dat" ,delimiter = ",") * c_s
print 'vr: ' , vth
vphi  = np.loadtxt("inpdata/velocity_phi.dat",delimiter = ",") * c_s
x_imp = np.loadtxt("inpdata/non_mrn/x.dat")


rel_delta_r_x =np.zeros(40)
print 'relative error between x and r grids in %'
for i,j in enumerate(np.searchsorted(r,x_imp)):
    rel_delta_r_x[i] = (r[j] - x_imp[i])/r[j]
    print i ,' : ', np.round(rel_delta_r_x[i]*100,2),'%'
```

```
print 'average error'
print np.round(np.mean(rel_delta_r_x)*100,2),'%'
print 'Standard deviation'
print np.round(np.std(rel_delta_r_x)*100,2),'%'

# create r-theta grid
r_gr,th_gr =np.meshgrid(r,th)
x =r_gr*np.sin(th_gr)
h =r_gr*np.cos(th_gr)

#calculate carthesian velocity components
vh = np.cos(th_gr)*vr - np.sin(th_gr)*vth
print 'vh: ' , vh
vx = np.sin(th_gr)*vr + np.cos(th_gr)*vth
print 'vx: ' , vx

# plot
density
 #spherical
plt.figure()
plt.contourf(r /AU ,th,np.log10(rho),400)
plt.xlabel('r [AU]')
plt.ylabel('$\Theta_a$')
plt.colorbar()

#overplot velocity
sp_arr=5  # spacing of velocity arrows
plt.quiver(r[::sp_arr] /AU ,th[::sp_arr],vr[::sp_arr,::sp_arr],vth[::sp_arr,::sp_arr])


plt.figure()
plt.contourf(x / AU,h / AU,np.log10(rho / rho_g),400)
plt.xlabel('Radius [AU]')
plt.ylabel('Height [AU]')
plt.ylim(0,25.0*r_g / AU)
plt.xlim(0,25.0*r_g / AU)
plt.clim(-4.0,0.5)
plt.colorbar().set_label('Density [$\rho$]')
#overplot velocity

plt.quiver(x[::sp_arr,::sp_arr]  / AU,
           h[::sp_arr,::sp_arr]  / AU,
           vx[::sp_arr,::sp_arr] ,
           vh[::sp_arr,::sp_arr] )


plt.show()

###


starting_point = np.searchsorted(r,x_imp) # np.arange(1,41,1)
print 'starting points of the streamlines'
print starting_point
print 'number of streamlines'
print np.size(starting_point)

frac         = 0.8
i_maxstep    = 150000
###


#Plot the streamlines

streams=[]
j=0
for i in starting_point:
    streams.append(0)
    streams[j] = streamlines.compute_stream(r,th,vr,vth,
                                            i,48,
```

```
                                            frac,i_maxstep,
                                            reverse = False)
    plt.plot(streams[j][0] / AU,streams[j][1] /AU )
    j=j+1



plt.xlabel('Radius [AU]')
plt.ylabel('$\Theta$ [rad]')
plt.draw()



#ok, now I know the streamlines. For simplicity we now go to cartesian coordinates
streams_cart = []
plt.figure()
for i in range(len(streams)):
    streams_cart.append(0)
    streams_cart[i]=[]
    streams_cart[i].append(0)
    streams_cart[i][0]=streams[i][0]*np.sin(streams[i][1])     #Streamline x
    streams_cart[i].append(0)
    streams_cart[i][1]=streams[i][0]*np.cos(streams[i][1])     #Streamline y
    plt.plot(streams_cart[i][0] / AU,streams_cart[i][1] / AU)



#overplot density
plt.contourf(x / AU,h / AU,np.log10(rho / rho_g),600)
plt.clim(-4.0,0.5)
plt.colorbar().set_label('Density [$\rho_g$]')
plt.ylim(0,40*r_g / AU)
plt.xlim(0,40*r_g / AU)
plt.xlabel('Radius [AU]')
plt.ylabel('Height [AU]')



#compute centrifugal acceleration everywhere
a_centr=vphi**2/(x)

#now for each streamline we compute the vector normal to the streamlines at each point
#then we compute the force along the streamline
#lastly, we compute for each position in the streamline the maximum grain size that is carried away with the wind
#and taking the minimum the one along all the streamline
tangent          = []
amax_streamlines = []            #2D Array
amax             = []            #1D Array
starting_radius  = []


for i in range(len(streams)):
    tangent.append(0)
    amax_streamlines.append(0)

    #tangent computation
    tangent[i]      = np.zeros((2,streams_cart[i][0].size-1))
    tangent[i][1]   = -(streams_cart[i][1][1:]-streams_cart[i][1][0:-1])     #dh
    tangent[i][0]   = -(streams_cart[i][0][1:]-streams_cart[i][0][0:-1])     #dx
    norm            = np.sqrt(tangent[i][0,:]**2+tangent[i][1,:]**2)         #dr
    tangent[i][0,:] = tangent[i][0,:]/norm
    tangent[i][1,:] = tangent[i][1,:]/norm

    #now compute the force (centrifugal acceleration)
    acentr_streamline = bilinear.interpolate2d_grid(r,
                                                    th,
                                                    a_centr.T,
                                                    streams[i][0],
                                                    streams[i][1])
```

```

    acentr_projected  = acentr_streamline[1:]*tangent[i][0,:]
    print "acentr_projected"
    print acentr_projected

    #e_R is the unity vector of the radial direction
    e_R      = np.zeros((2,streams[i][0].size))  #uniti vectoralong the direction
    e_R[0,:] = np.sin(streams[i][1])
    e_R[1,:] = np.cos(streams[i][1])

    agrav_streamline    = np.zeros((2,streams[i][0].size))
    agrav_streamline    = -(Grav*M_star/streams[i][0]**2)*e_R #gravitational acceleration vector
    agrav_projected     = agrav_streamline[0,1:]*tangent[i][0,:]+agrav_streamline[1,1:]*tangent[i][1,:] #projection alog thestreamline
    print "agrav_projected"
    print agrav_projected
    amax_streamlines[i] = np.zeros(streams_cart[i][0].size)

    # equation for drag force reversed for size
    amax_streamlines[i] = bilinear.interpolate2d_grid(r,th,rho.T,streams[i][0][1:],streams[i][1][1:]) * \
    (bilinear.interpolate2d_grid(r,th,vr.T,streams[i][0][1:],streams[i][1][1:])**2 + \
    bilinear.interpolate2d_grid(r,th,vth.T,streams[i][0][1:],streams[i][1][1:])**2) / \
    (-(agrav_projected+acentr_projected)*rho_dust)
    print " "
    print agrav_projected+acentr_projected
    print " "
    print "amax_streamlines[i]"
    print amax_streamlines[i]
    amax.append(np.min(amax_streamlines[i][np.where(amax_streamlines[i]>0)]))
    print 'amax: ', amax , i
    starting_radius.append(streams_cart[i][0][0])
    print 'starting_radius' , np.asanyarray(starting_radius) / AU

#plot the maximum grain size carried away in the wind as a function of cylindrical radius (base of the flow)
plt.figure()
plt.semilogy(np.asanyarray(starting_radius)/r_g,np.asanyarray(amax)*1e4,'.')
plt.xlabel('Cylindrical radius [r_g]')
plt.ylabel('Maximum grain size #$\mu$m]')


a_max = np.array(amax)
np.savetxt("a_max.dat",a_max)

tot1 = 0
for streamline in streams_cart:
    numb_points = len(streamline[0])
    tot1 += numb_points

x_str = np.zeros(tot1)  # will contain x coordinates of all streamlines
h_str = np.zeros(tot1)  # will contain h coordinates of all streamlines
a_str = np.zeros(tot1)  # will contain a_max values at all x,h coordinates of all streamlines

tot = 0

for index , streamline in enumerate(streams_cart):
    numb_points = len(streamline[0])
    grain_size  = np.ones(numb_points) * a_max[index]
    a_str[tot : tot + numb_points] = grain_size
    x_str[tot : tot + numb_points] = streams_cart[index][0]
    h_str[tot : tot + numb_points] = streams_cart[index][1]
    tot += numb_points

#carthesian grid
xi = np.linspace(0,40,num = 100) * r_g
hi = np.linspace(0,40,num = 100) * r_g

a_max_interp = mlb.griddata(x_str, h_str, a_str, xi, hi)
rho_interp   = mlb.griddata(x.flatten(), h.flatten(), rho.flatten(), xi, hi)
```

```
np.savetxt("a_max_r.dat" , a_max_interp)
np.savetxt("rho_r.dat" , rho_interp)

plt.figure()
plt.contourf(xi/r_g,hi/r_g,a_max_interp*1e4,400)
plt.xlabel('Cylindrical radius [r_g]')
plt.ylabel('Height [r_g]')
#plt.clim(0.2,2.0)
plt.colorbar().set_label('Maximum grain size [$\mu$m]')


#____________________________________________________________________________
#____________________________________________________________________________

''' NON_MRN GRAINSIZE DISTRIBUTION'''

#importing data for non mrn distribution

a = np.loadtxt("inpdata/non_mrn/a.dat")
x1D = np.loadtxt('inpdata/non_mrn/x.dat') /AU
z = np.loadtxt('inpdata/non_mrn/z.dat') /AU
rho_d_disk = np.loadtxt('inpdata/non_mrn/rho_d.dat')

# transforming x1d into a 2D array, so x and z have same shapes
x2D = np.zeros((40,60))
for i in range (40):
    for j in range (60):
        x2D[i][j] = x1D[i]


# computing total dust density for all grains (a<a_max) in the disk
rho_d_sum_amax = np.zeros((40,60))
for j in range (60):
    for i in range (40):
        for k in range (101):
            if (a[k] < a_max[i]):
                rho_d_sum_amax[i][j] += rho_d_disk[i*101+k][j]

# computing total dust density for all grains in the disk
rho_d_disk=np.reshape(rho_d_disk, (40,101,60))
rho_d_sum = np.sum(rho_d_disk,1)

#defining the midplain in the disk
mdpl_ind = 3      #midplain index
z0 = z[:,mdpl_ind]
print z0


#ratio of dust densities
ratio = np.zeros((40,60))
ratio = rho_d_sum_amax / rho_d_sum

ratio_1d = np.zeros(40)
for i in range(40):
    ratio_1d[i] = ratio[i][mdpl_ind]


ratio_str = np.zeros(tot1)  # will contain ratio values at all x,h coordinates of all streamlines

tot = 0

for index , streamline in enumerate(streams_cart):
    numb_points = len(streamline[0])
    ratio_along_the_str  = np.ones(numb_points) * ratio_1d[index]
    ratio_str[tot : tot + numb_points] = ratio_along_the_str
    tot += numb_points

ratio_1d_interp = mlb.griddata(x_str, h_str, ratio_str, xi, hi)
print np.shape(ratio_1d_interp)

```

```python
rho_d_non_mrn = np.zeros(np.shape(ratio_1d_interp))
rho_d_non_mrn = rho_interp * ratio_1d_interp / 100.0

np.savetxt('rho_d_non_mrn.dat',rho_d_non_mrn)

'''
PLOTS
'''

#plot the total dust density for all grainsizes in the disk
plt.figure()
plt.contourf(x2D,z,np.log10(rho_d_sum),100)
plt.plot(x1D,z0)   #midplane overplot
plt.xlabel('Radius [AU]')
plt.ylabel('Height [AU]')
plt.colorbar()

'''
#plot total dust density up to a_max
plt.figure()
plt.contourf(x2D,z,np.log10(rho_d_sum_amax),100)
plt.xlabel('Radius [AU]')
plt.ylabel('Height [AU]')
plt.colorbar()

#plot the ratio of dust densities in the disc
plt.figure()
plt.contourf(x2D,z,np.log10(ratio),100)
plt.xlabel('Radius [AU]')
plt.ylabel('Height [AU]')
plt.colorbar()
'''

#plot the ratio of dust densities outside the of disc
plt.figure()
plt.contourf(xi/r_g,hi/r_g,ratio_1d_interp,400)
plt.xlabel('Cylindrical radius [r_g]')
plt.ylabel('Height [r_g]')
#plt.clim(0.2,2.0)
plt.colorbar().set_label('ratio of dust densities outside the disc')


plt.show()
```

calculation of streamlines

This subroutine is responsible for calculation of streamlines. It is written in PYTHON, but it is not called by the main program directly. The CYTHON subroutine firs converts this code c code, which gets compiled in binary code, in order to speed up the calculation time.

```
from __future__ import division
import pyximport; pyximport.install()
import bilinear
import numpy as np
cimport numpy as np


#constants (in CGS)
#---------------------------------------------------------
m_p       = 1.6726231e-24        # proton mass in g
mu_bar    = 1.37125
gamma     = 5./3.
k_b       = 1.380658e-16         # Boltzmann constant in erg/K
rho_dust  = 1.
Grav      = 6.67259e-8           # gravitational constant in cm^3 g^-1 s^-2
M_sun     = 1.989e33             # mass of the sun in g
M_star    = 2.5*M_sun
AU        = 1.496e13             # astronomical unit in cm

u         = 1.660538782e-24      # atomic mass unit in g
c_s       = 1.0e6                # Speed of sound in ionized gas
PI        = 3.14159265359

#the imported Data is in scaled units
#Scale parameters (in CGS):

r_g    = ( Grav * M_star ) / (c_s**2)   # length scale
alpha2 = 2.6e-13                        # recombination coeff. for all states except ground state
m_H    = 1.008 * u                      # Hydrogen mass
m_mean = 1.35  * m_H                    # mean mass per hydrogen atom
PHI    = 1.0e43                         # ionizing flux ????
    #particle density scale
n_g    = 0.1*( (3*PHI) / (4*PI*alpha2*(r_g**3)) )**(1/2)
rho_g  = m_mean * n_g

#-----------------------------------------------------------------



def compute_stream(r,th,vr,vth,irin,ithin,frac,i_maxstep,reverse=False):

#    This function computes a single streamline by integrating velocity.
#    It uses a crappy Euler first-order integrator.


    if reverse:
        negative=-1.
    else:
        negative=1.
    #assumes regular grid
    dr           = r[1]-r[0]
    dth          = th[1]-th[0]

    r_current    = r[irin]
    th_current   = th[ithin]
    vr_current   = vr[ithin,irin]
    vth_current  = vth[ithin,irin]
    r_stream     = np.zeros(i_maxstep)
    th_stream    = np.zeros(i_maxstep)
    r_stream[0]  = r_current
    th_stream[0] = th_current
    np.seterr(divide = 'raise')

    for i in range(i_maxstep):
        vr_current  = bilinear.interpolate2d_grid(
            r,
            th,
            vr.T,
            np.array([r_current]),
            np.array([th_current])
```

```python
        )
        vth_current = bilinear.interpolate2d_grid(
            r,
            th,vth.T,
            np.array([r_current]),
            np.array([th_current])
        )
        dt          = negative*min(
            frac*dr/abs(vr_current),
            frac*dth*r_current/abs(vth_current)
        )

        th_current  = th_current+dt*vth_current/r_current
        r_current   = r_current+dt*vr_current

        # break conditions
        if r_current < r[0]:
            break
        if r_current > r[-1]:
            break
        if th_current < th[0]:
            break
        if th_current > th[-1]:
            break
        r_stream[i]  = r_current
        th_stream[i] = th_current

    return r_stream[0:i], th_stream[0:i]
```

bilinear interpolation

This subroutine is responsible for 2D interpolations. It is also written in Python, but it is also first compiled by the cython subroutine (see before) in binary language, in order to speed up the calculation time.

```
import pyximport; pyximport.install()
import numpy as np
cimport numpy as np

def interpolate2d_grid(x, y, Z, xnew, ynew):
    """Fundamental 2D interpolation routine

    Input
        x: 1D array of x-coordinates of the mesh on which to interpolate
        y: 1D array of y-coordinates of the mesh on which to interpolate
        Z: 2D array of values for each x, y pair
        xnew, ynew: arrays of points where the interpolation is wanted

    Output
        1D array with same length as points with interpolated values

    Notes
        Input coordinates x and y are assumed to be monotonically increasing,
        but need not be equidistantly spaced.

        Z is assumed to have dimension M x N, where M = len(x) and N = len(y).
        In other words it is assumed that the x values follow the first
        (vertical) axis downwards and y values the second (horizontal) axis
        from left to right.

        If this routine is to be used for interpolation of raster grids where
        data is typically organised with longitudes (x) going from left to
        right and latitudes (y) from left to right then user
        interpolate_raster in this module
    """


    #checks right shapes
    if x.ndim !=1:
        raise IndexError("x must have only 1 dimension!")
    if y.ndim !=1:
        raise IndexError("y must have only 1 dimension!")
    if Z.shape != (x.size,y.size):
        raise IndexError("Dimension of Z must be dimx*dimy!")
    if xnew.size != ynew.size:
        raise IndexError("xnew and ynew must have the same size!")

    #flattens xnew and ynew
    if xnew.ndim > 1:
        xnew=xnew.flatten()
    if ynew.ndim > 1:
        ynew=ynew.flatten()

#checks for points out of bounds
#    outside = np.count_nonzero(np.where(xnew<x[0]))+np.count_nonzero(np.where(ynew<y[0]))
+np.count_nonzero((xnew>x[-1]))+np.count_nonzero((ynew>y[-1]))
#  if outside > 0:
#  raise IndexError("Points out of boundary not implemented yet...")

    outside1=xnew<x[0]
    outside2=ynew<y[0]
    outside3=xnew>x[-1]
    outside4=ynew>y[-1]
    outside5=np.logical_or(outside1,outside2)
    outside6=np.logical_or(outside3,outside4)
    outside=np.logical_or(outside5,outside6)
    inside=np.logical_not(outside)


    # Find upper neighbours for each interpolation point
    idx = np.searchsorted(x, xnew[inside], side='left')
    idy = np.searchsorted(y, ynew[inside], side='left')

    # Get the four neighbours for each interpolation point
    x0 = x[idx - 1]
```

```
    x1 = x[idx]
    y0 = y[idy - 1]
    y1 = y[idy]

    z00 = Z[idx - 1, idy - 1]
    z01 = Z[idx - 1, idy]
    z10 = Z[idx, idy - 1]
    z11 = Z[idx, idy]

    # Coefficients for weighting between lower and upper bounds
    np.seterr(invalid='ignore')  # Ignore division by zero
    alpha = (xnew[inside] - x0) / (x1 - x0)
    beta = (ynew[inside] - y0) / (y1 - y0)

    # Bilinear interpolation formula
    dx = z10 - z00
    dy = z01 - z00
    z=np.zeros(xnew.size)
    z[inside] = z00 + alpha * dx + beta * dy + alpha * beta * (z11 - dx - dy - z00)
    #z[outside]=nan
    return z


```

B. Source Code for calculation of dust density distribution

This part of the code is responsible for calculation of dust density distributions, for both MRN and non MRN cases. It was written solely for the purposes of this thesis also in the programming language Python2.7.

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt


#constants (in CGS)
#---------------------------------------------------------
m_p    = 1.6726231e-24        # proton mass in g
mu_bar = 1.37125
gamma  = 5./3
k_b    = 1.380658e-16         # Boltzmann constant in erg/K
rho_d  = 1.
Grav   = 6.67259e-8           # gravitational constant in cm^3 g^-1 s^-2
M_sun  = 1.989e33             # mass of the sun in g
M_star = 2.5*M_sun
AU     = 1.496e13             # astronomical unit in cm

u      = 1.660538782e-24      # atomic mass unit in g
c_s    = 1.0e6                # Speed of sound in ionized gas
PI     = 3.14159265359

#the imported Data is in scaled units
#Scale parameters (in CGS):

r_g    = ( Grav * M_star ) / (c_s**2)   # length scale
alpha2 = 2.6e-13                        # recombination coeff. for all states except ground state
m_H    = 1.008 * u                      # Hydrogen mass
m_mean = 1.35  * m_H                    # mean mass per hydrogen atom
PHI    = 1.0e43                         # ionizing flux ????
    #particle density scale
n_g    = 0.1*( (3*PHI) / (4*PI*alpha2*(r_g**3)) )**(1/2)
rho_g  = m_mean * n_g

amin = 5e-7
amax = 0.1

#-----------------------------------------------------------------

x = np.linspace(0,40,num = 100)*r_g
h = np.linspace(0,40,num = 100)*r_g

a_max_r = np.loadtxt("a_max_r.dat")
rho_r   = np.loadtxt("rho_r.dat")
rho_d_non_mrn = np.loadtxt('rho_d_non_mrn.dat')

xi = np.linspace(0,40,num = 100) * r_g
hi = np.linspace(0,40,num = 100) * r_g

rho_dust = (rho_r/100.0)*(1-np.abs(np.sqrt((amin)/a_max_r)))

rho_dust_exact = ((rho_r/100.0)/(amin**(-0.5)-amax**(-0.5)))*(amin**(-0.5)-a_max_r**(-0.5))


plt.figure()
plt.contourf(x/AU,h/AU,np.log10(rho_r),200)


print rho_dust - rho_dust_exact

plt.figure()
plt.contourf(x/AU,h/AU,np.log10(rho_dust.T),200)
plt.xlabel('Radius [AU]')
plt.ylabel('Height [AU]')
plt.xlim(0,300)
plt.ylim(0,300)
#plt.clim(-26.,-22.)
plt.colorbar().set_label('Dust Density [g cm^-3]')

plt.figure()
plt.contourf(x/AU,h/AU,np.log10(np.abs(rho_dust_exact)),200)
```

```
plt.xlabel('Radius [AU]')
plt.ylabel('Height [AU]')
plt.xlim(0,300)
plt.ylim(0,300)
#plt.clim(-26.,-22.)
plt.colorbar().set_label('Dust Density [g cm^-3]')

plt.figure()
plt.contourf(xi/AU,hi/AU,np.log10(np.abs(rho_d_non_mrn)),200)
plt.xlabel('Radius [AU]')
plt.ylabel('Height [AU]')
#plt.xlim(0,300)
#plt.ylim(0,300)
#plt.clim(-26.,-22.)
plt.colorbar().set_label('Dust Density [g cm^-3]')

plt.show()
```

C. Supplement plots

This part of the code is responsible for calculation all supplementary plots used in this thesis, like figure 2.2. It was written solely for the purposes of this thesis also in the programming language PYTHON2.7.

```
from __future__ import division

import pyximport; pyximport.install()
import bilinear
import matplotlib.pyplot as plt
import matplotlib.mlab as mlb
import numpy as np
import streamlines

#constants (in CGS)
#---------------------------------------------------------
m_p       = 1.6726231e-24        # proton mass in g
mu_bar    = 1.37125
gamma     = 5./3.
k_b       = 1.380658e-16         # Boltzmann constant in erg/K
rho_dust  = 1.                   # g cm^-3
Grav      = 6.67259e-8           # gravitational constant in cm^3 g^-1 s^-2
M_sun     = 1.989e33             # mass of the sun in g
M_star    = 2.5*M_sun
AU        = 1.496e13             # astronomical unit in cm

u         = 1.660538782e-24      # atomic mass unit in g
c_s       = 1.0e6                # Speed of sound in ionized gas

#the imported Data is in scaled units
#Scale parameters (in CGS):

r_g    = ( Grav * M_star ) / (c_s**2)   # length scale
      'r_g [AU]'
      r_g
      r_g/AU
alpha2 = 2.6e-13                        # recombination coeff. for all states except ground state
m_H    = 1.008 * u                      # Hydrogen mass
m_mean = 1.35  * m_H                    # mean mass per hydrogen atom
PHI    = 1e43                           # ionizing flux s^-1

    #particle density scale
n_g    = 0.1*((3*PHI) / (4*np.pi*alpha2*(r_g**3)))**(1/2)

ng     = 2.8*(1e4)*((PHI/(1e41))**(0.5))*((M_star/M_sun)**(-1.5))


rho_g  = m_mean * n_g

#-----------------------------------------------------------------

#Import Data

r     = np.loadtxt("inpdata/radius.dat") * r_g
th    = np.loadtxt("inpdata/theta.dat" )
rho   = np.loadtxt("inpdata/density.dat"     ,delimiter = ",") * rho_g
      'rho' , np.shape(rho)
vr    = np.loadtxt("inpdata/velocity_r.dat"  ,delimiter = ",") * c_s
      'vr: ' , np.shape(vr)
vth   = np.loadtxt("inpdata/velocity_th.dat" ,delimiter = ",") * c_s
      'vr: ' , np.shape(vth)
vphi  = np.loadtxt("inpdata/velocity_phi.dat",delimiter = ",") * c_s
      'vphi: ' , np.shape(vphi)

# create r-theta grid
r_gr,th_gr =np.meshgrid(r,th)
x =r_gr*np.sin(th_gr)
h =r_gr*np.cos(th_gr)

v = np.sqrt(vr**2+vth**2)
vh = np.cos(th_gr)*vr - np.sin(th_gr)*vth
vx = np.sin(th_gr)*vr + np.cos(th_gr)*vth
```

```
sp_arr=5
plt.figure()
plt.contourf(r /AU ,th,np.log10(rho/rho_g),400)
plt.xlabel('r [AU]').set_size('x-large')
plt.ylabel('$\Theta$ [rad]').set_size('xx-large')
cbar = plt.colorbar()
cbar.set_label('Density [$n_g$]',size=20)
cbar.ax.tick_params(labelsize=20)
plt.quiver(r[::sp_arr] /AU ,th[::sp_arr],vr[::sp_arr,::sp_arr],vth[::sp_arr,::sp_arr])

plt.figure()
      np.searchsorted(th,np.pi/4) , ' , pi/4'
      np.searchsorted(th,np.pi/6) , ' , pi/6'
      np.searchsorted(th,np.pi/3) , ' , pi/3'
      np.searchsorted(th,np.pi/2) , ' , pi/2'
plt.plot(r/AU,v[0,:]/1e6)
plt.plot(r/AU,v[17,:]/1e6)
plt.plot(r/AU,v[25,:]/1e6)
plt.plot(r/AU,v[33,:]/1e6)
plt.legend(("$\Theta$=0","$\Theta$=$\pi$/6","$\Theta$=$\pi$/4","$\Theta$=$\pi$/3"),loc=4)
plt.xlabel('r [AU]').set_size('xx-large')
plt.ylabel('$V_w$ [km/s]').set_size('xx-large')


#importing data for density weighted grainsize plots
a_max_r = np.loadtxt("a_max_r.dat")
rho_r   = np.loadtxt("rho_r.dat")                  # gas density distribution
rho_d_non_mrn = np.loadtxt('rho_d_non_mrn.dat')    #non mrn dust density
                                                     #distribution
rho_dust_exact = np.loadtxt('rho_dust_exact.dat')  #mrn dust density
                                                     #distribution

#coordinate grid
x = np.linspace(0,40,num = 100)*r_g
h = np.linspace(0,40,num = 100)*r_g


    i    range(100):
        j    range(100):
           (i > 10)&(j<10):
               (rho_d_non_mrn[i][j]>1e-23):
                rho_d_non_mrn[i][j]=0
    i    range(100):
        j    range(100):
           (i < 10)&(j>10):
               (rho_d_non_mrn[i][j]>1e-23):
                rho_d_non_mrn[i][j]=0

plt.figure()
plt.contourf(x/AU,h/AU,np.log10(np.abs(rho_d_non_mrn)),200)
plt.xlabel('Radius [AU]').set_size('xx-large')
plt.ylabel('Height [AU]').set_size('xx-large')
plt.colorbar().set_label('Dust Density [$g$ $cm^{-3}$]',size=20)



'''MRN CASE'''

#a_max_r        = np.nan_to_num(a_max_r)
a_max_r        = a_max_r.flatten()
rho_dust_exact = np.nan_to_num(rho_dust_exact.flatten())
rho_dust_exact = rho_dust_exact.flatten()

plt.figure()
plt.semilogx(rho_dust_exact,a_max_r*1e4, linestyle='n.an', marker='.',
     markerfacecolor='b')

plt.figure()
plt.semilogx(rho_dust_exact,a_max_r*1e4,linestyle='nan', marker='.',
     markerfacecolor='b')
```

```
'''NON MRN CASE'''
rho_d_non_mrn = np.nan_to_num(rho_d_non_mrn)
rho_d_non_mrn = rho_d_non_mrn.flatten()


plt.semilogx(rho_d_non_mrn ,a_max_r*1e4,linestyle='nan', marker='.',
     markerfacecolor='g')
plt.xlabel('dust density, logarithmic scale [$g/cm^3$]').set_size('xx-large')
plt.ylabel('grainsize [$\mu$m]').set_size('xx-large')

plt.figure()
plt.semilogx(rho_d_non_mrn ,a_max_r*1e4,linestyle='nan', marker='.',
     markerfacecolor='b')
plt.xlabel('dust density, logarithmic scale [$g/cm^3$]').set_size('xx-large')
plt.ylabel('grainsize [$\mu$m]').set_size('xx-large')
plt.show()
```

Bibliography

Armitage, P. J. (2011). Dynamics of Protoplanetary Disks. *Annual review of astronomy and Astrophysics*, 49:195–236.

Birnstiel, T., Klahr, H., and Ercolano, B. (2012). A simple model for the evolution of the dust population in protoplanetary disks. , 539:A148.

Birnstiel, T., Ormel, C. W., and Dullemond, C. P. (2011). Dust size distributions in coagulation/fragmentation equilibrium: numerical solutions and analytical fits. , 525:A11.

Font, A. S., McCarthy, I. G., Johnstone, D., and Ballantyne, D. R. (2004). Photoevaporation of Circumstellar Disks around Young Stars. *The Astrophysical Journal*, 607:890–903.

Owen, J. E., Ercolano, B., and Clarke, C. J. (2011). The imprint of photoevaporation on edge-on discs. *Monthly Notices of the Royal Astronomical Society*, 411:1104–1110.

Perrin, M. D., Duchêne, G., Kalas, P., and Graham, J. R. (2006). Discovery of an Optically Thick, Edge-on Disk around the Herbig Ae Star PDS 144N. *The Astrophysical Journal*, 645:1272–1282.

Takeuchi, T., Clarke, C. J., and Lin, D. N. C. (2005). The Differential Lifetimes of Protostellar Gas and Dust Disks. *The Astrophysical Journal*, 627:286–292.

Acknowledgements

I would like to thank my advisers Barbara and Giovanni. Both of them were very supportive, during last month’s and always answered my all questions with great patience. I have to give Giovanni spatial thanks, for that great amount of time he invested in me while explaining everything about Phython syntax (and many other things) from scratch. I also enjoyed the ”astro-ph” meetings and I want to thank the whole group for that interesting time, and their feedbacks. Finally I want to thank Silvan and Tobi, all credits for finding this wonderful work group goes to them.

List of Figures

Selbstständigkeitserklärung

Hiermit versichere ich,

dass ich diese Bachelorarbeit zum Thema: "Effekt des Staubteilchen-Wachstums, auf den Staubdgehalt des photoevaporativen Flusses der protostelaren Scheiben" selbstständig verfasst habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht.
Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

München, August 1, 2017

Ort, Datum

Unterschrift