INTRODUCTION AND CODE SUMMARY
Subroutines for a lime-extension that will simulates polarized line radiative transfer. Polarization of the molecular states is assumed to come only from anisotropy in the total intensity. An assumption that lends its veracity from the generally weakly polarized molecular emission lines that are observed. The low fractional polarization lacks the power to polarize the molecular states.

To model anisotropy in the local radiation field we map it with higher directional sampling than the output of the lime-code. We use the output of the lime radiative transfer code: a set of molecular state densities per (weighted but randomly placed) node, where each node has associated properties like temperature and velocity. For each node, we investigate a large number of rays that are directed outwards and ray-traced (using the above mentioned properties) inwards to the center of the node. The result of the inward ray-tracing is a precise mapping of the anisotropy of the radiation field at the location of the node. The direction of the rays are determined by a Gaussian quadrature (for the theta-angles) and an equal spacing for the phi-angles. Alternatively, a sampling can be chosen that incorporates a central stellar source from where high intensity light is coming, and that has a solid angle dependent on the node position and stellar radius.

The inward ray-tracing of the direction sampling of the radiation field yields the J^0_0 (isotropic) and the J^2_0 (anisotropic) parts of local radiation field, and are computed for every transition relevant to the molecule of interest. These numbers are inserted in the statistical equilibrium equations (SEE) for a molecule in the non-coherence case (strong magnetic field). The SEE are set up as given in Eq. (7.17-18) in L&L04. We note that we use the extension for collisions that are given in Eq. (7.101) in L&L04. We point out that for the collisional rates, we only have the isotropic rates available. Higher order tensorial expansion elements should also be calculable from quantum dynamical methods, but are not available as of yet. To balance the loss of polarization this simplification gives, we neglect de-polarization from elastic collisions.

In order to solve the SEE, we assume steady-state in the state-populations, and set the time-derivative of the populations zero. The lime-parent code follows a similar procedure---however only accounts for the isotropic populations. In the case that we compute N polarized populations, we have N separate, but interdependent equations, that can be expressed in matrix-form as
0 = A*p
where p is a vector of polarized densities and A is an NxN matrix with elements determined by the SEE. Just as in the Hazel code (Ramos 2008), we replace the first line of A (A(1,:)) with the normalization requirement in matrix form:
\sum_i sqrt([j_i])\rho_0^0 (j_i) = 1. L-84 Eq.(56)
With that constraint on the populations, the matrix equations become
[1;0;0;...] = A*p,
and can be solved by a matrix-inversion to obtain the polarized populations.

We then have acquired the polarized populations for all the levels in our molecule (or atom) of interest. In order to later ray-trace (a)

transition(s), we require polarization propagation coefficients. These are obtained from the polarized population coefficients using the Eqs. (33)-(36) from Landi 1984. They have to be computed for a particular angle theta between the local magnetic field and the ray-tracing direction. With these coefficients obtained, we can ray-trace the polarized radiation through the simulation.

The main program is divided up in two executables. The first executable is a program that performs the inward ray-tracing and solves the polarized SEEs for each node in the simulation. It outputs the polarized populations (k=0 and k=2) and has the option to output the radiative/ collisional rates as a check. The second executable uses the polarized populations to perform the polarized ray-tracing of the investigated region. It outputs the intensity-maps in Stokes I,Q and U.

A bash-script is added in which the input parameters can be defined. The script will then prepare the proper input files and run the executables of PORTAL. Note that PORTAL is parallelized with Open-MP, and that it is advised to run this program in parallel mode. Apart from the input file, PORTAL requires the location of the LIME populations.pop file as well as the grid.vtk file. Furthermore, PORTAL requires the LAMDA data file and a data file on the dust-opacity.

INSTALLING PORTAL

The script 'make_portal' includes the installation directions to compile the PORTAL code. We make use of the LAPACK libraires (http://www.netlib.org/lapack/), OpenMP libraires (https://www.openmp.org/) and the WIGXJPF library (http://fy.chalmers.se/subatom/wigxjpf/). These libraries have to be installed before compiling PORTAL.

The installation of PORTAL yields two executables: "main_portal.x" and “finish_portal.x". PORTAL is designed so that the polarized populations are saved after a “main_portal.x" calculation. Subsequently, one has the liberty to ray-trace the solution in any inclination angle, transition, or velocity-channel one so desires. Ray-tracing step is done with the "finish_portal.x" executable.

RUNNING PORTAL

PORTAL has an accompanying 'run_portal.sh' bash-script that prepares the input files that have to be given to PORTAL to run properly. The script is thoroughly commented. PORTAL requires files to be inputted:

1) input_file

The structure of the input file for "main_portal.x" and "finish_portal" are slightly different:

```
input_main:
<empty line>
star_mode T_star R_star X_star Y_star Z_star
nth nph nvel
b_rad b_tor b_pol b_dip
kmax extra_info
max_ne_temp n_depth
<empty line>
EOF

star_mode               -- if 1, do ray-tracing with an internal (or
external) star
T_star, R_star          -- temperature and radius star
X_star Y_star Z_star    -- position star
nth nph nvel            -- inward ray-tracing parameters (st: nph=nth=32
nvel=10)
b_rad b_tor b_pol b_dip -- magnetic field character of simulation ex:
b_rad=1, rest 0,
                           fully radial mag field. If b_dip not 0, then
dipole magnetic field
kmax                    -- k-truncation (st: kmax=6)
extra_info              -- do all the rates have to be outputted (st.
extra_info=0)
max_ne_temp n_depth     -- parameter-length parameters -- (st.
max_ne_temp=10000, n_depth=10000)

input_finish
<empty line>
ntrans
l1_1 l1_2 l1_3 ... l1_ntrans
l2_1 l2_2 l2_3 ... l2_ntrans
<empty line>
ninc
th_inc_1 th_inc_2 ... th_inc_ninc
ph_inc_1 ph_inc_2 ... ph_inc_ninc
<empty line>
star_mode T_star R_star X_star Y_star Z_star
b_rad b_tor b_pol b_dip
kmax extra_info
pix_mod im_pix im_size s_d
max_ne_temp n_depth_rt
<empty line>
EOF
```

2) location of the output map

3) population.pop file

population.pop file has to have the following structure:

x y z vx vy vz nH2 Tkin Tdust Doppler_b mol_ab gasiidust conv_flag pops_1 pops_2 ... pops_N

4) grid.vtk

5) molecular_lamda_file

6) dust_file (standard "jena_thin_e6.tab")

The run_portal.sh script prepares all the necessary files for the simulation.

INDIVIDUAL SUBROUTINES AND SRC FILES
We will now give some information on the subroutines located in this map in the files
./src/*.f90
Extensive commenting is available inside these files.

---

main_para.f90
Main file containing "program main" that calls the subroutines to read in LIME output, and call the inverse ray-tracing to compute the alignment of all of the energy-levels at all nodes in the simulation

---

eat_lime.f90
eat_lime_dim:
subroutine to read in the relevant dimensions of the lime-programs. Dimensions will be used to allocate the arrays. first step in reading in the data.

eat_lime_no_delaunay:
subroutine to read in the data of the lime-programs. dimensions read in before and are used to allocate the arrays.

---

legendre_rule.f90
These routine computes all the knots and weights of a Gauss quadrature formula with a classical weight function with default values for A and B, and only simple knots. Original FORTRAN77 version by Sylvan Elhay, Jaroslav Kautsky. FORTRAN90 version by John Burkardt.

---

main_sub.f90
trace_node:
main driver of the node-specific polarized population-calculations

---

inverse_rt.f90
inverse_rt:
for a certain node construct all the relevant rays using a Gaussian quadrature for the theta angles and an equal spaced gridding for the phi angles. The ray-paths are constructed with the trace_out subroutine. With the acquired ray-paths, we subsequently raytrace them to the node center. The product now is Stokes-I for certain theta and phi angles. These are used directly in the integration over the angles. N.B.!: theta and phi are gauged with respect to the magnetic field direction of the central node and a canonical axis that is use to construct the x-axis (freely choosable because phi is integrated out).

trace_out:
subroutine that traces the path from central node to an outward source for a certain photon-direction. Subroutine returns a list of the nodes it passes (including for how long a photon will be associated with a certain node) in depth_list

ray_in:

using the ray-trace depth_list of all the nodes with associated distances, and using information about these nodes (temp,vel,densities) to construct and solve the radiative transfer towards the central node

---

pop_eqs.f90

t_abs_const:
subroutine to compute the constants for absorption t_A from L&L04 Eqs.(7.20a)

t_stim_const:
subroutine to compute the constants for stimulated emission t_S from L&L04 Eqs.(7.20c)

t_spont_const:
subroutine to compute the constants for spontaneous emission t_E from L&L04 Eqs.(7.20b)

r_abs_const:
subroutine to compute the constants for absorption r_A from L&L04 Eqs.(7.20d)

r_stim_const:
subroutine to compute the constants for stimulated emission r_S from L&L04 Eqs.(7.20f)

r_spont_const:
subroutine to compute the constants for spontaneous emission r_E from L&L04 Eqs.(7.20e)

setup_stateq:
subroutine to setup up the statistical equilibrium equation matrix. Uses the constants from the aforementioned subroutines, and properly multiplies them with the (an)isotropic intensities (see Eqs. (7.17-18) in L&L04. Also incorporates collisions. This subroutine first reads through a list of transitions, and determines which population-elements are coupled. It then uses this list to fill in the matrix elements for de-excitations to level, excitations to level, and all (de-)excitations to the level itself. The eventual matrix will be inverted to obtain the polarized populations.

---

em_coefs.f90

comp_emcoefs:
subroutine that uses the polarized populations from two states to compute the polarized emission coefficients between them.

get_rhovec:
subroutine that obtains the rho-vector of a certain node by setting up the SEE and subsequently solving them. Also has the option to obtain the rates of collisions, spontaneous and stimulate/absorption emission events.

---

para_finish.f90

program to do the second part of the PORTAL calculation: using the aligned populations, obtain a map of the polarization of an astro-physical region.

---

raytrace.f90

polarize_prop:
subroutine to do a propagation of polarized radiation over a cell.

trace_out_raytrace:
subroutine that traces from a starting point on the sphere to the observer on the other side of the sphere, listing the nodes it passes in depth_list

ray_in_raytrace:
using the ray-trace list of all the nodes with associated distances, using information about these nodes (temp,vel) and the earlier computed polarization parameters to ray-trace through the simulation

---

miscellaneous.f90

check_neigh:
subroutine to check which node is closest to the photon. Uses the Delaunay neighbour-list of node the photon had previously been associated to.

find_dir:
Find the photon-direction vector that is associated to a certain theta and phi angle. Theta and phi are with respect to the magnetic field direction (bvec) and a certain canonical direction (gvec) that is the same for all the cells in the simulation.

find_angles:
subroutine to find the angles from the direction vector

rotmat:
Constructs the rotation-matrix for a rotation of angle th around the vector nv.

two_mat_vec_prod:
A matrix-matrix-vector product that: A*B*c = d

setup_count_array:
subroutine that sets up the count-matrix. The matrix that will keep track of what count-number in the stateq-matrix, will correspond to what state (l,j,k)

matvec_prod:
compute the product A*b = c

black_body:
black_body radiation

make_AC:
subroutine to construct the AC matrix --- the matrix containing all te relevant information for the SEE equations

all_transitions:
find all unique upper-to-lower transitions

rot_stokes:
subroutine to rotate the stokes vectors to the appropriate coordinate system of the node. Subroutine assumes that the ray-direction is along the z-axis, and the canonical axis is along the x-axis

rot_stokes_back
subroutine to rotate the stokes vectors back to the appropriate global coordinate system. Subroutine assumes that the ray-direction is along the z-axis, and the canonical axis is along the x-axis

dipole_magfield:
subroutine to define the global dipole magnetic field, and assign each node a magnetic field direction.

global_magfield:
subroutine to define the global magnetic field, and assign each node a magnetic field direction.

dust_to_opacities:
subroutine to convert the jena_thin_e6 tables to the correct opacities for the dust for each transition-frequency

rotate_all_save:
rotate axis system to ray-tracing axis system by rotating all the vector quantities.

adj_list_no_delaunay:
algorithm to construct the neighbour list from the delaunay triangulation in a memory efficient manner.

check_neigh_fast:
subroutine to compute the distance to the next neighbour

pos_quadratic:
subroutine that returns the minimal and positive solution to the quadratic equation

legendre_all:
subroutine to set up the integration of the sphere, using a legendre-discretization for theta and using a variable number of phi-points (weighed with sin(th)

star_all:
subroutine to set up the integration of the sphere, while respecting the definite solid angle of anb extra source of radiation.

circle_points:
subroutine that divides a number of points roughly evenly on a circle.