style=

RX-DMFIT User's Guide

August 25, 2017

Contents

1 Introduction **2**

2 Examples **2**
2.1 example1.cpp . . . 2
2.2 example2.cpp . . . 2

3 Constants **3**

4 External Libraries **3**
4.1 DarkSUSY . . . 3
4.1.1 Darksusy.h . . . 3
4.2 GNU Scientific Library . . . 3

5 Target Class **4**
5.1 Target.h . . . 5
5.1.1 Redshift/Distance . . . 5
5.1.2 Magnetic Field . . . 5
5.1.3 Dark Matter Profile . . . 5
5.1.4 Diffusion . . . 6
5.1.5 vLUT . . . 7
5.1.6 GLUT . . . 7
5.1.7 Member Functions . . . 8
5.2 Particle.h . . . 9
5.3 DMprofile.cpp . . . 9
5.4 bfield.cpp . . . 9
5.5 dist.cpp . . . 9
5.6 emissivity.cpp . . . 9
5.7 flux.cpp . . . 10
5.8 greens.cpp . . . 10

5.9 pIC.cpp . . . 11
5.10 psyn.cpp . . . 11
5.11 diffusion.cpp . . . 11
5.12 surfaceBrightnessProfile.cpp . . . 11

6 Run Files **12**
6.1 Run.h . . . 12
6.2 calc_sv.cpp . . . 12
6.3 Run.cpp . . . 12

List of Tables

1 Magnetic field model selections . . . 6
2 Dark matter profile selection . . . 6
3 Diffusion handling selections . . . 7
4 Diffusion coefficient selection . . . 7

1 Introduction

RX-DMFIT (Radio and X-ray - DMFIT) is a tool for calculating the predicted multi-wavelength emission profile and spectra from dark matter annihilation. Particulary, RX-DMFIT calculates the emission due to synchrotron, inverse Compton scattering, and pion decay to gamma-rays, as well as providing dark matter constraints for user provided radio or X-ray data. A wide range of customizable astrophysical and particle parameters are included and important astrophysical effects are incoroprated into the analysis, including diffusion, energy loss processes, and magnetic field mdeling.

2 Examples

Two example files are included that demonstrate some basic functionality of RX-DMFIT.

2.1 example1.cpp

In example1.cpp we calculate the emissivity profiles and spectra from DM annihilation in the Coma galaxy cluster. As previously demonstrated, diffusion effects in clusters are typically negligible [4, 1] and are neglected in this analysis. Multiple annihilation channels and magnetic fields are considered, and constraints on the DM particle cross-section are placed using radio data from [5]. This code can be run with the terminal command:

```
make example1
./example1
```

2.2 example2.cpp

In example1.cpp we calculate the emissivity profiles and spectra from DM annihilation in the Draco dwarf spheroidal galaxy. In dSphs, diffusion effects have a considerable impact that should not be negelected [4], and is thus included in this analysis. This code can be run with the terminal command:

```
make example2
./example2
```

3 Constants

Constant values used in RX-DMFIt are declared in `Constants.h` and initialized in `Constants.cpp`. units and descriptions can be found in either file.

4 External Libraries

4.1 DarkSUSY

The electron/positron and gamma injection spectrum per dark matter annihilation event, dN/dE_{inj} are determined using the Fortran package DarkSUSY [3]. RX-DMFIT was originallly developed using DarkSUSY v5.1.2 and has not been tested with other versions. The DarkSUSY packagev can be obtained at: `http://www.darksusy.org/`

4.1.1 Darksusy.h

The functions that interface with the DarkSUSY library are located in `Darksusy.h`.

`void dsinit_()`:
Fortran routine that initializes DarkSUSY and must be called each run in order to make use of the DarkSUSY library. Note that calling this function more than once during a run of RX-DMFIT will cause an error and end the program.

`double dshayield_(double *mwimp, double *emuthr, int *ch, int *yieldk, int *istat)`:
The Fortran routine that provides the injection spectrum from DarkSUSY. This routine is called in `darksusy(double Ep)` and `darksusy_gamma(double Egamma)` which pass `E` and `Egamma` as the `*emuthr` argument. `*yieldk` determines which DM annihilation product is computed (e.g. electron/positrons or gamma-rays).

`double darksusy(double Ep)`:
Calculates the electron/positron injection spectrum from DarkSUSY for energy `Ep`.

`double darksusy_gamma(double Egamma)`:
Calculates the gamma-ray injection spectrum from DarkSUSY for energy `Egamma`.

4.2 GNU Scientific Library

Numerical integration in RX-DMFIT is carried out using the GNU Scientific Library (GSL) [2]. Two integration methods rae utilized: QNG non-apadtive Gauss-Kronod (QNG) and QAGS adaptive integration with singularities (QAGS). The QNG integration is preferred where possible as it is considerably quicker. In Integrils such as the diffusion and emissivity integrals the QNG method is typically sufficient, although numerical artifacts can occasionally appear. In these integrals the QNG integration is the default, but the method can be easily changed QAGS by changing a boolean value. For example if we wanted to perfrom the diffusion integral in `Target::diffusion( double E, double r)` (in `diffusion.cpp`), we would change the code:

```
if(true){
    gsl_integration_qng (&F, E, p.mx, 0, 1e-3, &result, &error, &size);
}
else{
    gsl_integration_workspace * w = gsl_integration_workspace_alloc (1000);
    gsl_integration_qags (&F, E, p.mx, 0, 1e-3, 1000, w, &result, &error);
    gsl_integration_workspace_free (w);
}
```

so that it reads:

```
if(false){
    gsl_integration_qng (&F, E, p.mx, 0, 1e-3, &result, &error, &size);
}
else{
    gsl_integration_workspace * w = gsl_integration_workspace_alloc (1000);
    gsl_integration_qags (&F, E, p.mx, 0, 1e-3, 1000, w, &result, &error);
    gsl_integration_workspace_free (w);
}
```

For more information about the different methods, see
`https://www.gnu.org/software/gsl/manual/html_node/Numerical-Integration.html`

5 Target Class

The Target class contains all of the astrophysical parameters considered in RX-DMFIT and the majority of functions used. Note that in order to be compatible with the GSL integration this is a static class, and thus only one instance can be used at any given time. When creating an instance of this class an instance of the Particle class is also created (see `Particle.h`), which contains the DM model parameters such as M_χ, $\langle\sigma v\rangle$, and the branching ratios. The member variables are declared and described in `Target.h` and initialized in `Target.cpp` with values based on the NCC model from [4]. The Target member variables are:

Distance/Size

z
distance
rh

B-field

BB
B0
rcore
beta
eta

DM Profile

DM
rhos_NFW
rs_NFW
alpha_NFW
rhos_Ein
rs_Ein
alpha_Ein

Energy Losses

bsynch
bIC
bICSL
bbrem
bcoul
nele

Starlight Model

r_b
n_s
b_n
r_d

Diffusion

SD
DD
gamma
db
D0
imNum

v Lookup Table (vLUT)

vsize
vscale
vlookup

Green's Lookup Table (GLUT)

n_r
n_rootdv
rootdv_max
GLUT

Most of the parameters are straightforward or discussed in [4] (e.g. B_0, r_h, etc.) so we will only discuss a few parameters of note here.

5.1 Target.h

5.1.1 Redshift/Distance

```
static double z ;
static double distance;
```

The redshift `z` is best used for distant objects such as galaxy clusters. If a redshift value is provided, the comoving distance will be calculated. Alternatively, for some systems such as local group ddwarfs using a distance in terms of kpc if preferred. In this case, setting `z == 0`, will result in RX-DMFIT using the `distance` value for calculations.

5.1.2 Magnetic Field

```
static int    BB;
. . .
static double beta;
static double eta;
```

The `BB` value selects the magnetic field model, with `BB == 0` selecting the basic exponential model and `BB == 1` selecting the beta-fit model. Additionally, a constant magnetic field can be used by selecting the beta-fit model and and setting `eta == 0` or `beta == 0` To summarize:

5.1.3 Dark Matter Profile

```
static int DM;
```

Params.	$B(r)$
`BB == 0`	$B_0 \exp -r/rc$
`BB == 1`	$B_0 \left[1 + \left(\frac{r}{rc}\right)^2\right]^{-1.5\beta\eta}$
`BB == 1` , `beta` or `eta == 0`	B_0

Table 1: Magnetic field model selections

The `DM` value selects the dark matter density profile, where `DM == 0` uses the NFW profile and `DM == 1` uses the EInasto profile. Note that RX-DMFIT now includes an α parameter in the NFW profile which was not discussed in [4]. In summary:

Params.	$\rho_\chi(r)$
`DM == 0`	$\rho_s / \left[(r/r_s)^\alpha (1 + r/r_s)^{3-\alpha}\right]$
`DM == 1`	$\rho_s \exp\left\{-2/\alpha \left[(r/r_s)^\alpha - 1\right]\right\}$

Table 2: Dark matter profile selection

5.1.4 Diffusion

```
static int SD;
```

The role of the `SD` value is twofold: it is used to select whether or not to include diffusion effects, as well as selecting hos to handle the Green's function calculation. Analytical calculations of emissions from DM annihilation can become quite computationally expensive, so in order to speed up the calculations we have made use of a lookup table for the $v(E)$ integral(vLUT), as well as a two-dimensional lookup table for the Green's functions (GLUT). These tables will be discussed in more depth in sections **??** and **??**, respectively, and for now we will focus only on the `SD` options. The option `SD == 0` is the no spatial diffusion (NSD) option, where no Green's function is needed (see for example equation 4 in [6]). In the cases where `SD == 1, 2, 3` diffsuion is included, however the Green's function is handled differently in each case.

Setting `SD == 1` selects the actively populating GLUT. In this case, each time a call is made to `Target::ddiffusion(double Ep, void * params)` the program checks to see whether the corresponding Green's function value (i.e.`Target::greens(double r, double root_dv)`) value in the GLUT is empty (i.e., = 0). If it is empty, a value is calculated and stored. If there is a value already stored, that value is used. This is typically the fastest and most precise method of running RX-DMFIT with diffusion included.

Setting `SD == 2` selects the option that does not use the GLUT at all. In this case, each time a call is made to `Target::ddiffusion(double Ep, void * params)`, the Green's function value is calculated directly. This is by far the slowest method and is not appropriate for calculating final spectra, but it can be useful for troubleshooting or checking that you are not getting inaccurate or unexpected results.

Setting `SD == 3` selects the prepopulated GLUT. This option requires running the function `Target::createGLUT()` after assigning parameters and prior to running Run functions. In this case, `Target::createGLUT()` fills in all the values in the GLUT (meaining n_r*n_rootdv calls to `Target::greens(double r, double root_dv)` before any other calculations. Then when performing your analysis any calls to `Target::ddiffusion(double Ep, void * params)` read the Green's function value from the GLUT. This method performs the emission calculations slightly more quickly than the actively populated method, with the caveat that the `Target::createGLUT()` operation is much more time-consuming. To summarize:

Params.	Diff.
`SD == 0`	No Spatial Diffusion
`SD == 1`	Actively populate GLUT
`SD == 2`	No GLUT used
`SD == 3`	Prepopulate GLUT

Table 3: Diffusion handling selections

```
static int DD;
```

The `DD` value select the form of the diffusion coefficient, where `DD = 0` selects the simplified power law in energy, and `DD = 1` incorporates the degree of uniformity of the magnetic field.

Params.	D(E)
`DD == 0`	$D_0 E^\gamma$
`DD == 1`	$D_0 E^\gamma \left(d_b^{2/3} / B_{avg}^{1/3} \right)$

Table 4: Diffusion coefficient selection

```
static int imNum;
```

When placing boundary conditions on the Green's function for the diffusion equation, the image charge method is used (see Appendix A of [1]). In theory, this involves taking an infinite sum of image charges, i.e. $G(r, \Delta v) = \sum_{n=-\infty}^{n=\infty} G_n(r, \Delta v)$ where G_n is the Green's function prior to placing the boundary condition. In practice, we only need enough image charges so that $G_n \approx 0$. Therefore we have $G(r, \Delta v) = \sum_{n=-imNum}^{n=imNum} G_n(r, \Delta v)$ where $G_{imNum} \approx 0$. RUNGREENSTERM

5.1.5 vLUT

```
static int vsize;
static double vscale;
static std::vector<double> vlookup;
```

The vector `vlookup` is where the pre calculated $v(E)$ values are stored. `vsize` is initialized in `Target`cpp. to a fairly large value of 100000. The `vscale` value is calculated when `vlookup` is created.

5.1.6 GLUT

```
static int n_r;
static int n_rootdv;
static int rootdv_max;
static std::vector< std::vector<double> > GLUT;
```

The Green's funcction values are tored in the `GLUT`. Note that the Green's function can be expressed as a function of two arguments: $G(r, \sqrt{\Delta v})$. when the `vLUT` is created, the maximum allowed value for $\sqrt{\Delta v}$ is calculated to determine `rootdv_max`. The values `n_r` and `n_rootdv` determine the size of the `GLUT`. They are both set to 1001 by default. In some cases, such as small `rootdv_max` values it is better to decrease the `n_rootdv` in order to avoid numerical errors. Also, in particular when using `SD == 3`, decreasing `n_rootdv` can be used to reduce computing time. The `n_r` value can aso be changed in order to reduce the computation time, however note that when calculating the profiles (e.g. `runjsynch()`, `runjIC_CMB()` etc.) the radial values run from r_h/n_r to r_h.

5.1.7 Member Functions

The member functions of the Target class are:

Distance/Size

```
ddist(z , *params)
dist()
rconst(rcm)
```

B-field

```
bfield_model (r)
bfield_model (r, *params)
Bmu()
```

Energy Losses

```
bloss(E, r)
bloss(E)
```

DM Profile

```
DM_profile(r)
```

Greens

```
D(E)
dv(E, *params)
v(E)
root_dv(Ep, vE)
dgreens_term(rp, *params )
greens_term(ri, r, root_dv, rh)
greens(r, root_dv)
create_vLUT()
clearGLUT()
createGLUT()
```

Diffusion

```
ddiffusion(double Ep, *params)
diffusion(E, r)
elec_spect(E, r)
```

Psynch

```
fff(x)
dpsynch(theta, *params )
psynch(E, r, nu)
```

PIC

```
G(eps, E_gamma, boost);
IC_cross_Section(eps, E_gamma, E);
CMB_bbSpectrum(eps);
SL_Spectrum(eps);
SL_profile(r);
dpIC_CMBeps, *params)
pIC_CMB(nu, E)
dpIC_SL(eps, *params)
pIC_SL(nu, E)
```

Emissivity

```
djIC_CMB(E, *params)
jIC_CMB(nu, r)
djIC_SL(E, *params)
jIC_SL(nu, r)
djsynch(E, *params)
jsynch(nu, r)
```

Surface Brightness Profile

```
dI_synch(E , *params)
I_synch(nu, theta)
I_synch(nu, theta, beam)
```

Fluxes

```
dsIC_CMB(r, *params)
sIC_CMB(nu)
dsIC_SL(r, *params)
sIC_SL(nu);
dssynch(r, *params)
ssynch(nu);
ssynch(nu, r)
dspion(r, *params)
spion(nu)
```

These functions will be discussed in depth in the appropriate sections below.

5.2 Particle.h

This header conatains the defintion for the Particle class and initializes mass, cross-section, and branching ratios. Note that branching rations are initialized to 0, and should sum to unity. The Particle class is a sub class of the Target class, and a Particle instance is created when a Target instance is created. Contains the member variables:

```
double mx;
double sv;
double BR_WW;
double BR_ee;
double BR_mumu;
double BR_tautau;
double BR_bb;
```

5.3 DMprofile.cpp

Contains `double Target::DM_profile(double r)`.

5.4 bfield.cpp

Contains:

```
double Target::bfield_model (double r)
double Target::bfield_model (double r, void * params)
double Target::Bmu()
```

Here we overload the bfield function to have compatibilty with the GSL integration, used in Bmu() to compute an average magnetic field strength. The argument `r` is in cm, and the outputs for each function are in μG.

5.5 dist.cpp

Contains:

```
double Target::ddist(double z  , void * params)
double Target::dist()
double Target::rconst(double rcm)
```

`dist()` is the integral over `z` of `ddist(double z  , void * params)`, i.e. the comoving distance. `rconst(double rcm)` returns a raidus adjusted for radio observationsso that the region of interest is smaller than the beam radius. The units for the output of `dist()` and `rconst()` are both cm.

5.6 emissivity.cpp

Contains:

```
double Target::djIC_CMB(double E , void * params)
double Target::jIC_CMB(double nu, double r)
double Target::djIC_SL(double E , void * params)
double Target::jIC_SL(double nu, double r)
double Target::djsynch(double E , void * params)
double Target::jsynch(double nu, double r)
```

These are the emissivity integrands and integrals. Note however that the factor $\langle\sigma v\rangle\,\rho^2(r)/(2M_\chi^2)$ has been pulled out of the source term. In other words, in order to calculate an inverse Compton emissivity value, you would need,

```
Target::p.sv/(2*pow( Target::p.mx , 2 ))
    *pow(Target::DM_profile(r) , 2) * Target::jIC_CMB(nu, r )
```

This output would be in GeV/cm^3/s/Hz.

5.7 flux.cpp

Contains:

```
double Target::dsIC_CMB(double r, void * params)
double Target::sIC_CMB(double nu)
double Target::dsIC_SL(double r, void * params)
double Target::sIC_SL(double nu)
double Target::dspion double r, void * params)
double Target::spion(double nu)
double Target::dssynch(double r, void * params)
double Target::ssynch(double nu)
double Target::ssynch(double nu, double r)
```

Integrated flux density integrands and integrals. The `nu` arguments are in Hz, and the flux outputs are in GeV/cm^2/s/Hz. The function `Target::ssynch(double nu, double r)` is intended for use with the adjusted radius (see `Target::rconst(double rcm)`).

5.8 greens.cpp

```
double Target::D(double E)
double Target::bloss(double E)
double Target::dv(double E , void * params)
double Target::v(double E)
double Target::root_dv(double Ep,  double vE)
double Target::dgreens_term(double rp, void * params )
double Target::greens(double r, double root_dv)
void Target::create_vLUT()
void Target::clearGLUT()
void Target::createGLUT()
```

This file contains all of the calculations for the Green's function, as well as functions to create the vLUT and GLUT. Note that `create_vLUT()` is called in each of the run functions. `createGLUT()` is only needed for the case where `SD == 3`. The clearGLUT(). function should be called whenever any of the parameters that effect the Green's function are changed, which includes almost all of the parameters excpet for `distance, mx, sv`, all of the branching ratios and the starlight model parameters. In the RX-DMFIT code, the spatially independent form of the energy loss equation, `bloss(E)`, is only used in calculating `v(E)`. The function `greens(r, root_dv)` integrates `dgreens_term(rp, *params)` and includes bouth the QNG and QAGS integration options as discussed in section 4.2.

5.9 pIC.cpp

```
double Target::G(double eps, double E_gamma, double boost)
double Target::IC_cross_Section( double eps, double E_gamma, double E)
double Target::CMB_bbSpectrum(double eps)
double Target::SL_Spectrum(double eps)
double Target::SL_profile(double r)
double Target::dpIC_CMB(double eps, void * params )
double Target::pIC_CMB(double nu, double E)
double Target::dpIC_SL(double eps, void * params )
double Target::pIC_SL(double nu, double E)
```

Calculations for the IC power, including integrands and integrals. `G(eps, E_gmma, boost)` While part of the power expression, the starlight spatial profile given by `SL_profile(r)` is pulled out of this integral and is instead called in the flux integrand, `dsIC_SL(nu)`. Note that the power is in units of GeV/s/Hz.

5.10 psyn.cpp

```
double fff(double x);
double dpsynch(double theta, void * params );
double psynch( double E, double r, double nu);
```

Calculations for the synchrotron power, including integrands and integrals. `fff(x)` and `(x)` are defined in equations 2.11 and 2.10 of [4]. Again the power is in units of GeV/s/Hz.

5.11 diffusion.cpp

```
double darksusy_gamma (double Egamma)
double darksusy (double Ep)
double Target::ddiffusion(double Ep, void * params)
double Target::diffusion( double E, double r)
double Target::bloss(double E , double r)
double Target::elec_spect(double E, double r )
```

The functions `darksusys(Ep)` and `darksusy_gamma(Egamma)` call the functions that interface with the Fortran DarkSUSY package, and provide the $e^{\pm}$ and gamma-ray spectra respecitvely. `darksusys(Ep)` is called in `ddifusion(Ep, *params)` as well as the Green's function in accordance with the selection for `SD`. The integral is performed in `diffusion(E, r)` and the final $e^{\pm}$ spectrum (`elec_spect(E, r)`) is obatined by dividing by the spatially dependent energy loss equation, `bloss(E, r)`.

5.12 surfaceBrightnessProfile.cpp

NOTE: These functions are not complete and may need modifications.

```
double dI_synch(double E , void * params);
double I_synch(double nu, double theta);
double I_synch(double nu, double theta, double beam);
```

These functions compute the surface brightness profile of the target. For `I_synch(nu, theta)`, `nu` and `theta` are in units of Hz and kpc, repectively and the output is in mJy/arcmin^2. In `I_synch(nu, theta, beam)`, `nu` and `theta` are in again in units of Hz and kpc with `beam` in units of arcmin. Here the output is in mJy/beam.

6 Run Files

6.1 Run.h

Cross Sections

```
min_flux(r)
sv_RadioPredict(nu)
calc_svIC_CMB(nu, flux_obs )
calc_svSynch(nu, flux_obs )
```

Emissivity

```
runjsynch(mx, nu)
runjIC_CMB(mx, nu)
runjIC_SL(mx, nu)
```

Surface Brightness Profile

```
runI_synch(mx, nu)
runI_synch_mx(beam, r,  nu)
```

SED & Fluxes

```
runSED_IC_CMB(mx)
runSED_IC_SL(mx)
runSED_IC(mx)
runSED_pion(mx)
runSED_synch(mx)
runSED(mx)
runFlux(mx, fluxType)
```

Exclusion Curves

```
runExCurveSynch(nu, flux_obs)
runExCurveIC(nu, flux_obs)
```

6.2 calc_sv.cpp

```
double calc_svIC_CMB(double nu, double flux_obs )
double calc_svSynch(double nu, double flux_obs )
double min_flux(double r)
double sv_RadioPredict(double nu)
```

6.3 Run.cpp

```
double min_flux(double r);
double sv_RadioPredict(double nu);
double calc_svIC_CMB(double nu, double flux_obs );
double calc_svSynch(double nu, double flux_obs );

void runjsynch(double mx, double nu);
void runjIC_CMB(double mx, double nu);
void runjIC_SL(double mx, double nu);

void runI_synch(double mx, double nu);
void runI_synch_mx(double beam, double r,  double nu);

void runSED_IC_CMB(double mx);
void runSED_IC_SL(double mx);
void runSED_IC(double mx);
void runSED_pion(double mx);
void runSED_synch(double mx);
void runSED(double mx);
```

```
void runFlux(double mx, int fluxType);
void runExCurveSynch(double nu, double flux_obs);
void runExCurveIC(double nu, double flux_obs);

void runGreensTerm(double r, double root_dv);
```

References

[1] S. Colafrancesco, S. Profumo, and P. Ullio. Multi-frequency analysis of neutralino dark matter annihilations in the Coma cluster. A&A, 455:21–43, August 2006.

[2] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Michael Booth, and Fabrice Rossi. *GNU Scientific Library Reference Manual - Third Edition*. Network Theory Ltd., 3rd edition, 2009.

[3] P. Gondolo, J. Edsjö, P. Ullio, L. Bergström, M. Schelke, and E. A. Baltz. DarkSUSY: computing supersymmetric dark matter properties numerically. J. Cosmology Astropart. Phys., 7:008, July 2004.

[4] A. McDaniel, T. Jeltema, S. Profumo, and E. Storm. Multiwavelength Analysis of Dark Matter Annihilation and RX-DMFIT. *ArXiv e-prints*, May 2017.

[5] E. Storm, T. E. Jeltema, S. Profumo, and L. Rudnick. Constraints on Dark Matter Annihilation in Clusters of Galaxies from Diffuse Radio Emission. ApJ, 768:106, May 2013.

[6] E. Storm, T. E. Jeltema, M. Splettstoesser, and S. Profumo. Synchrotron Emission from Dark Matter Annihilation: Predictions for Constraints from Non-detections of Galaxy Clusters with New Radio Surveys. *ArXiv e-prints*, July 2016.