Program Summary: Tracking Magnetic Monopoles Through the Galaxy

Felix Feist

GOALS / TIMELINE

due: approx.. april 24 – 8 months from today.

goal for 9/26: finish runge-kutta alg, PLUS an euler to test their consistency.

PAPER TO DO:

-\*discussion of mass and other chosen properties

-\*discussion of error sources and bounds. sources:

-sigfigs, python numerical errors

-GMF model

-mathematical approximations, including nonrelativistic

-\*add: assumes r is never precisely 0 because, well, it makes infinities. and the xfield is weird.

-\*figure numbers, equation numbers

-\*make mathematica things in word

-\*change image of diagram to computer image (photoshop?)

-\*read through as if unaware of anything and check for clarity (but also eliminate over-explaining)

-\*add healpix section with diagrams

-\*make all hats bold

-\*add section about nonreal gamma and v>c for too small timestep

-error: we got divide by zero for r-rp=0, even when z=/= 0 because of python rounding errors. check for the possibility of other such errors.

-explain as if to an undergrad, but then at end move the more obvious to appendix

-redo eqn for disk field for 3<r<5, making it just phihat, and explaining later what that is.

-read about radiation and whether or not it’s relevant to the analysis

OTHER TO DO:

-check if I’m running extraneous processes on ria

-check formulas in word doc

-run for negative charge (antimonopole), 2 charge, 1/3 or 2/3 charge, different masses, etc

-relativistic version: beware of not using copyto for acceleration etc, acc might change when you change vel or something

-error: we got divide by zero for r-rp=0, even when z=/= 0 because of python rounding errors. check for the possibility of other such errors.

-change units to cgs Gaussian

-consider simple situations with predictable results and test program output on those.

-asyncpy parallel computing

-read about parity and CP violation, they are fundamental to nature!

-rounding error must be checked. see Wikipedia for “machine epsilon”

-optimization: if necessary to optimize, have a “smart check” for the disk regions: only check adjacent regions to the one which I was in previously. Note that a large timestep (or changing timestep) can mess this up.

goals:

I. map all of the possible conditions of the earth-hitting monopoles to their positions and velocities on the 20kpc sphere.

1. Is the magnetic field even important? Does it change their velocities significantly at all? AKA do we even need this computational business?

-check for global vars in a local function problem http://stackoverflow.com/questions/10851906/python-3-unboundlocalerror-local-variable-referenced-before-assignment

-if computation is necessary: numerical recipes book.

2. find errors. stepsize error, computer rounding error. human error in building the program – test for cases in which you know something about the outcome.

3. check for other monopole charges

II. find out if there is a decent chance of having monopoles bound in the galaxy

III. Making an actual paper stuff

Bring up w Farrar, or other questions:

**-What is the intention in making a plot of where monopoles came from? What is it going to help?**

**-Even with the upper bound of mass, the final velocity ends up as 0.3c just on one nonrelativistic run. Seems I should, in general, run the relativistic version**

**-are we sure about using 16.7 for r\_s, not the mean value? Refer to plot in paper, pg 7**

**-statistical analysis: what kind of error are we looking at due to our uncertainty in the bfield?**

**-Newq:**

**-Wick[6]: “We emphasize…relativistic.” Let me make sure I understand their argument: If monopoles pass through extragalactic sheets, the regions with largest energy changes, they will gain about 1e14GeV in a coherence length. Monopoles are presumed to have gone through these sheets at least once (is that because they are presumed to have been created in the early universe during a phase transition?), and so their energies are presumed to be of that order.**

**-equation 9, where does it come from, what’s the deal?**

**-what portion of a galaxy is empty space? I ask to know whether collisions would be relevant.**

**-why are flux units 1/(cm^2 s sr)? why is the sr in there too (I know the definition of the sr, that is not the issue)**

**-GZK cutoff not applicable for monopoles? (see notes, wick paper). somewhat of a comment, not a question.**

**-equation 2.1… it says below what those two symbols are but I don’t know enough to say from that what their orders of magnitude would be and thus what the mass bound of equation 2.4 is**

**-wondering: is it worth it to find the potential of the magnetic field? I was going to do so to find out which areas have a negative difference in potential great enough that a relativistic monopole of some mass entering the galactic sphere would be slowed to nonrelativistic speeds before hitting the earth. It would be better than just running the program for different directions, I would actually have a full answer as to which parts of the galactic sphere have enough of a potential difference. I could then also run the program to see which parts of the area on the galactic sphere would actually result in earth collisions. BUT if I did find that the difference in potential was NOT great enough to change the velocity of a particle by a significant amount, the paths would be nearly straight lines. Or would they be? Just because the energy difference wouldn’t be much, does that mean anything about the *direction* of the particles as they hit the earth? What I am trying to find out is: are all of the particles just going to go in a straight line anyway? (this would make our analysis kinda pointless).**

**-what about the magnetic field for rho <1 kpc? Since the black hole is there I would think that there are some extremely strong magnetic fields that we couldn’t ignore in that region. Is there an analysis of that region?**

**Introduction**

The program takes input parameters and initial conditions, and tracks a particle backward in time, tracing out the path it took through the galaxy for a specified distance. It then plots the path in 3D along with a yellow dot where the sun is, and a blue dot at the center of the galaxy. The 3D plot is interactive and can be rotated by dragging. Below the plot of the particle trail is a plot of its kinetic energy as a function of time. The program works for particles with arbitrary properties, but by default it has theoretically likely properties of a magnetic monopole. The galactic magnetic field is given by Jansson, Farrar (2012).

The input parameters are magnetic charge, mass, initial position and velocity, timestep, and distance to track the particle. The program uses base units of distance in kpc, time in s, current in A, and mass in kilograms. All other units are built from these. In other words, all units are SI units except those with factors of distance. The kinetic energy, however, is plotted in GeV.

A few assumptions were made in designing this program:

1. There are no collisions – space is empty. While this is not true, it is somewhat unlikely for a particle to collide with an object, and if it did, its path from that point on would be uninteresting. It is also worth noting that realistically, the only collisions that could take place are collisions with objects which act as a source of magnetic monopoles, because the particle was tracked *backwards* in time.

2. The magnetic monopoles have no electric charge.

3. All forces are considered negligible except for the magnetic force.

4. The particle cannot decay during its travel.

In addition, some mathematical assumptions were made in the theory, in order to make the program possible. Most, if not all of these assumptions rely on a small timestep. To test whether the chosen timestep is small enough that the assumptions make negligible difference, divide the timestep by some factor and see if the final position is significantly different than it was with the initial timestep.

The mass of the default particle is chosen to be the lower mass bound of a magnetic monopole from Wick (2002) – 40TeV/c2.

The charge is one elementary unit of magnetic charge, given by the dirac quantization condition. This is further described in the Theory section.

By default, the particle starts at the position of the earth. The galactic center is not plotted by default because if the particle is only tracked a short distance, showing both the center and the trail would require the window to be so large that the trail cannot be seen in detail.

Because of the limitations on numbers of significant digits in python, the Lorentz factor is precisely equal to 1 until . In this case the kinetic energy is calculated using .

The program currently uses a constant timestep. But if one were to vary the timestep so that when the velocity is great, the timestep gets smaller, it would prevent large changes in distance in a single step and increase accuracy.

**Theory**

**I. Monopole Charge and Mass**

The dirac quantization condition, with magnetic charge in units of Ampere meters, gives

Here, qb is the elementary magnetic charge. Converting to Ampere kiloparsecs yields:

(we use 8 digits, the accuracy to which is known)

In Gaussian units, the quantization condition, from wikipedia, is

Or,

As for the units, I get:

(based on wikipedia’s definition of the erg and statC)

**II. Relativistic Particle Dynamics**

For comparison, here is the non-relativistic derivation, in one dimension, with the assumption that force is constant over the interval :

To track the particle backwards, we just reverse the sign of the equations for and , which are the only equations appearing in the program itself.

We now consider the relativistic case. We use 3-vectors, as coordinate transformations are not necessary. All quantities are classical (so, for example, ).

The force on a magnetic monopole in a magnetic field is:

I assume that this holds true, even relativistically, because of the way the Coulomb law was used in Griffiths (pg 524). However, here, F is the “classical” force, , as opposed to the nicely-transforming .

Thus we have

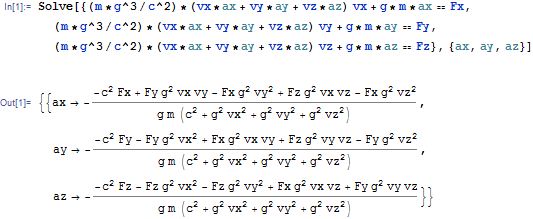
Newton’s second law in the form does not hold. Instead, (still with classical variables and three-vectors) we use

With , its derivative is:

Plugging this into the force yields:

The above equation reduces to if .

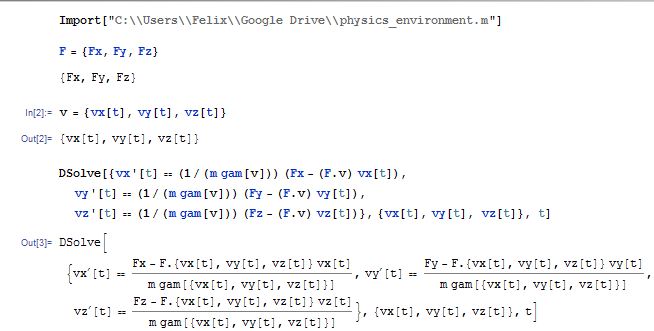
The equations for the three components of yield a system of three equations and three unknowns ( and ). Mathematica solves for the three accelerations:



dividing by γ^2 and expanding out the definition of γ:

**C:\Users\Felix\Downloads\CodeCogsEqn (15).gif**

It seems that this system of differential equations is not solvable; mathematica knows no solution, even with a constant force.



We now have solved for , and integrating component-by-component yields , the change in velocity over the time interval . In order to do so, we make the assumption that our timestep is small enough that the 6 dependent variables of **a**, namely **F** and **v**, are constant over the interval, and thus the acceleration is as well. We do so with the x-component below:

With the above assumption, we integrate, and solve for

We then integrate again to find Δx:

In our program, then, we repeat the following process:

1. Calculate the force

2. Using the force and initial velocity, calculate the acceleration

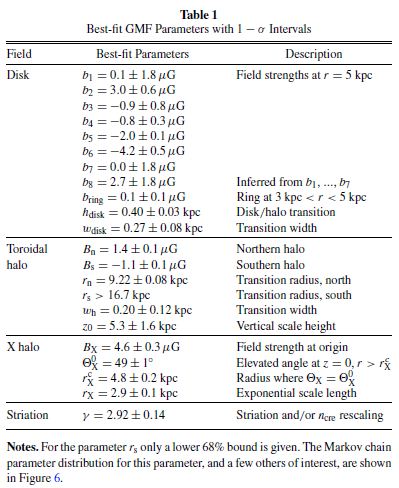
3. Using the above expressions for and , update the positions and velocities to their new values at

**III. The magnetic field from Jansson, Farrar (2012): Summary and Calculations**

Using Faraday rotation measures, it was possible to extrapolate an approximate value of the magnetic field throughout the galaxy. The field everywhere is a sum of three functions: The disk component. the toroidial halo component, and the out-of-plane component (referred to as the X field because of its shape). The paper mentions a fourth component (the striated field), which was not used in this program. The different components are defined for different regions. Where the regions overlap, the field is a sum of the components.

This paper uses right-handed coordinate systems. They are centered at the galactic center. In rectangular coordinates, the sun is at x=-8.5kpc, and the z axis is perpendicular to the galactic plane. In cylindrical, r is the distance from the galactic center axis, and z is the same as in rectangular coordinates. The paper uses a convention in which the cylindrical coordinate extends from the negative x-axis, but this program adopts a convention in which extends from the positive x-axis. Thus, we replace all instances of in the paper by . From this point, our convention for the angle will be written , and the paper’s convention will be written .

The magnetic field depends on a variety of best-fit parameters. The table of parameters from the paper is below:



In a sphere of radius kpc, and outside the cylinder with , the field is defined to be 0. Below is a summary of its components in the nonzero region.

**IIIa. The Disk Component:**

The disk field is 0 for .

For , the disk field is azimuthal and takes the form

C:\Users\Felix\Downloads\CodeCogsEqn (5).gif

is a function which transitions the disk field into the halo field. The halo field is multiplied by , and the disk field is multiplied by . Its definition is

When , the disk component is a piecewise function, dependent on the “region” in the galaxy wherein the function is evaluated. The regions are defined as follows:

8 logarithmic spirals on the x-y plane form boundaries (thus creating 8 regions). The equations of the spirals take the form NO THEY DONT

Here, is the opening angle of the logarithmic spiral (it is the same for each spiral).

\*(make this a bottom comment) Note: In the paper it’s:

But that is an error.

is the position at which the spiral crosses the negative x-axis. The values it takes on are 5.1, 6.3, 7.1, 8.3, 9.8, 11.4, 12.7, and 15.5, in kpc. Regions are numbered such that Region 1 is between the functions with and , region 2 is between and , and regions 3-8 follow.

The disk component of the magnetic field in region j is

C:\Users\Felix\Downloads\CodeCogsEqn (4).gif

C:\Users\Rick\Downloads\CodeCogsEqn.gif

Where the direction is:

C:\Users\Rick\Downloads\CodeCogsEqn (1).gif

C:\Users\Rick\Downloads\CodeCogsEqn (4).gif

C:\Users\Rick\Downloads\CodeCogsEqn (2).gif

C:\Users\Rick\Downloads\CodeCogsEqn (3).gif

is a constant parameter which takes on one of the 8 values listed in the table at the beginning of section III.

In rectangular coordinates, the disk component in region j is



The disk field is nonzero in the region with . In this region it is azimuthal and takes the form

C:\Users\Felix\Downloads\CodeCogsEqn (5).gif

where

C:\Users\Felix\Downloads\CodeCogsEqn (12).gif

**IIIb. The Toroidial Halo Component**

The toroidial halo component is a simple piecewise function:

C:\Users\Felix\Downloads\CodeCogsEqn (6).gif

It is defined to be 0 for r = 0, where its direction is not well-defined.

**IIIc. The X-Field Component**

The X-field has no azimuthal component, giving it the structure for which it was named:

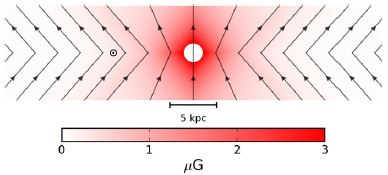


Diagram above from Jansson, Farrar (2012). This is the xz cross section, so and it includes the sun at x=-8.5kpc, but the field lines the same for any .

The field is given as a function of , the radius at which a field line hits the x-y plane. Before , the field lines are variable in slope with elevation angle . Afterwards, the slope remains constant at a value of .

Thus, in the constant elevation region,

and in the varying elevation region,

The respective fields are

C:\Users\Felix\Downloads\CodeCogsEqn (6).gif

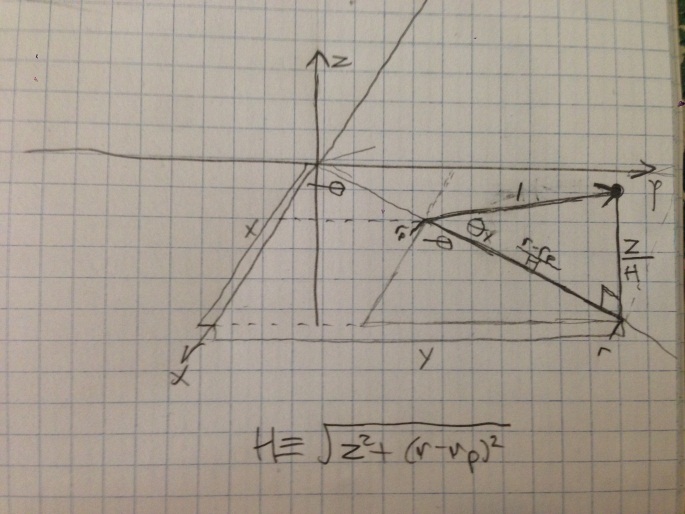
where C:\Users\Felix\Downloads\CodeCogsEqn (4).gif is a unit vector defined by the field direction in the X-field image. (give figure number), which, for , comes out to

C:\Users\Felix\Downloads\CodeCogsEqn.gif

C:\Users\Felix\Downloads\CodeCogsEqn (5).gif

For , C:\Users\Felix\Downloads\CodeCogsEqn (4).gif is straight upward in z. This was not in the paper, but was chosen because a similar and slightly better-fit xfield function has a bfield straight upward in z. For r=0, C:\Users\Felix\Downloads\CodeCogsEqn (4).gif normally would reach a singularity but is defined as C:\Users\Felix\Downloads\CodeCogsEqn (10).gif for z>0 and -C:\Users\Felix\Downloads\CodeCogsEqn (10).gif for z < 0. Note that for r=0, where , is still well-defined (just divide the formula for rp by r).

eq# (above) can be derived from the diagram below, where .



The program checks which equation to use for by checking the inequality

The inequality divides the two regions: if true, use the varying elevation region.

This concludes all three components of the magnetic field.

**Method of ODE Solving**

We use a 5th order Runge-Kutta with an adaptive timestep method and coefficients suggested by Cash and Karp, taken from Numerical Recipes in C (page 740).

The general procedure:

dt = guess step

//startloop

allowedDelta[] = maximum allowed error per step allowed in each component of **r** and **v**, estimated beforehand

Delta[] = actual errors in each of the components after taking the guess step dt. //this is estimated by doing rungekutta5 – rungekutta4 = estimateDelta.

if (any of the actual errors (Delta) are greater than the allowed errors) {

#retake the step

dt <- 0.9\*dt\*(allowedDelta[theWorstOffenderBy%]/Delta[theWorstOffenderBy%])^0.25

xnew, vnew = rk4&5(xold,vold,dt) //rk4&5 is a combination of the terms in 4 and 5 that yields 6th order.

else {

//set dt in preparation for next step:

dt = 0.9\*dt\*(allowedDelta[theWorstOffenderBy%]/Delta[theWorstOffenderBy%])^0.2

}

…repeat the process until finished.

The following discussion approximates the total error in position and then in velocity. Some relevant variables:

Ntot : total # of steps in a path Nsofar : # steps so far in a path

BL : approximate upper bound for B field = 3 μG (the subscript L stands for Large)

sL  : approximate upper bound for path lengths = 30kpc. ssofar : path length so far

L : actual path length of a traversal

Δtot : desired total error (absolute) Δstep : desired error per step (absolute)

v0  : initial velocity vfinal : velocity at end of path

We do not subscript Δtot to specify whether we are speaking of the distance estimate or the velocity estimate: in the distance error discussion, it means the distance error, and in the velocity error discussion, it means the velocity error. We believe it will be clear to the reader which is meant.

We now estimate the error in distance traveled:

The total error is desired to be less than one percent:

//the error should actually be the error in each x coordinate. Ignore this bit. the dx one is done below.

We approximate :

Because we do not know the actual value of L before the algorithm is complete, we use a larger-than-expected value of the total path length in place of L, giving an upper bound:

By transitivity of inequality in equations (\*elephant) and (\*elephant), and replacing all L with sL,

Solving for Δstep gives

Thus,

…where 30kpc is typically greater than the total distance of a path, in order to over-estimate the error and put us below one percent total error. The steps taken and distance traveled will be continually updated as the path is traversed. **If, after the traversal, the path length is greater than 35kpc, we redo the calculation with the actual path length in this formula.** IMPLEMENT THIS WHEN YOU RUN IN LARGE QUANTITIES.

Now I’m redoing it because I shouldn’t have used arclength last time.

We construct the final error in the x-position after a path is finished, :

where   is the average stepsize. We substitute the approximation . In other words, the proportion of the current number of steps to the final number of steps is the same as the proportion between the final and current arclengths traveled, .

y

We provide an over-estimate for to give a safe error: for this we choose double the radius of the galaxy, . \*footnote If the program finds its total length to be more than , we re-run the program with this approximated as 1.5 times the actual arc-length traversed.

We want our error to be less than .125kpc, which is 0.1% of the circumference of our 20kpc sphere. Outside of a 20kpc sphere, we stop tracking the monopoles. Solving for the step size gives

To address all three directions, we simply require that all three step sizes be less than this quantity.

REDOING AGAIN because I don’t like that it is not constant. It adds more machine work.

We wish to find the maximal error allowed in one step of the algorithm, requiring our final position to be off by only . is the radius of the region in the milky way in which we track. As a first step, we solve for the error in our x coordinate, and then we generalize to all positions and velocities. The following variables are used:

Δ1 is the average error in one step in the x direction, it can be negative.

Δtot is the x error at the end of the trip, it can be negative

x1 is the average stepsize in x, it can be negative

xtot is the final x coordinate

is the average timestep

is a typical x-velocity. We use a value which is attained near the halfway point of the trajectory.

By definition, and.

Solving for gives

I will eliminate x1 using (this is exactly true, I have checked it) where dtbar is the average timestep throughout the run and can be negative. First I prove the equation and then state the updated equation for Δ1:

(by a definition of avg velocity)

(divide by N: now, - boom.)

so,

To eliminate , consider the following: Our algorithm is 5th order, so the error in each step is approximately equal to the 6th order term:

I will save the problem of finding the coefficient for later, leaving Δ1 defined as . I now eliminate dt by solving for :

Solving for gives

We want the fractional error in each coordinate to be at most F=0.001. So we require that . Because could be any x-coordinate on the sphere, and because

To simplify further, we would like to find an expression for . We choose to approximate it as a typical x-velocity, . This is reasonable because the magnetic field strength has no large outliers, so a typical field strength will lie close to the average.

To find use Wx = ΔKEx. “The x-component of the work?” you say? Well, the work can be separated component-wise even though it is not a vector with our constant force approximation. Consider this proof:

W = ΔKE

Moving the x-dependence all to one side makes the other a function of only y and z:

Since the above statement is true for all values of x, y, and z, both sides must be constant if they are to be equal. So we have:

But this is also true for all values along the trajectory. Consider the point at which the particle has not yet moved. Then both the terms on the left vanish, so the constant must equal zero, giving us

We proceed to use this formula to solve for :

To explain the left hand side: The force on a monopole due to a magnetic field is qB. We estimate that a typical velocity will occur after the particle is displaced from its starting position by a distance , but if all directions traveled are equidistant, the x-component of the distance traversed is only .

Isolating gives:

With solved for, we can now return to equation \*^\*:

This, along with equation \*^\*, is the (estimated) expression for the maximum allowed error in one step in the x direction. N

Ok so last time I was using xtot which is not a good measure… it could be anything from -20 to 20. And if it’s near 0 that’s drastically different… bad.

What if I use arc length at the beginning instead? But for now I will leave N in there.

By definition, .

We want the error in each coordinate to be a very small percentage of the r=20kpc sphere. Call that fraction of tolerance F, so that the error allowed is

Thus,

Now, how do we find N, the number of steps in the algorithm? We can’t. But we can find a “safe” N by choosing a large value for N so that the allowed error per step is small. This safe N can be estimated as:

…where s is the arclength traversed so far. The estimate is based on the algorithm ending at latest when the particle has traversed an entire diameter of the milky way. Then we have simply:

From here, a reasonable algorithm would be:

Run the programming assuming the actual arclength traveled will be less than .

If the program runs longer than , cancel and double the threshold to 4. Repeat as long as necessary. In addition, double the threshold for all initial conditions in a sphere surrounding the “failed” run in phase space, so that runs are not wasted on initial conditions which are likely to cause a re-run anyway.

Now we must repeat the procedure for the velocity. We seek the average error per step allowed given the constraint that we want a final error of less than E. Just like before,

with the constraint that and , this gives

But what should we choose as E? The problem really highlights a problem with the backtracking approach: we want to be able to make the assumption that the incoming velocity of monopoles is a particular number, and find the percent of the initial phase space with that initial velocity which hits the earth. But with backtracking, we give a velocity to start with and end up with a finishing (or starting – in either case, earlier in time) velocity which is not of our choosing. It doesn’t suit the problem at hand. So for now, I will choose an arbitrary E – that is the lesser of one one-hundredth the speed of light or of the initial velocity.

Equation \*^\* (for ) must be positive anyway. We are looking for an absolute error to compare to. So the outside of the square root was omitted; we will just take the abs value of this anyway. But what if this is greater than c? We just substitute c in that case. The error does not have limiting properties near c, it is essentially linear.

Looking at equation \*^\*, it is clear that if we make our upper limit for the error Δ1 small, we will be more likely to reject a step (we reject if it is greater than Δ1) and so our error will be on the conservative side. Since Δ1 is monotone increasing with , we choose the negative sign within the square root. With this, equation \*^\* reads

where

This is the approximate error in x per step, which we use as an upper bound for our actual error per step by setting F=0.001. Note that it depends only on initial conditions and is constant for a given run.

We also need a measure of the error in the components of velocity. The rationale for the error per step in vx, which we call Δ1v, is the same as that in x up to equation \*^\*:

Now, , if we impose a fractional final error of . We make the linear approximation that ; in other words, the velocity increased by the same amount in the second half of the journey (the second half in time, not distance). This gives:

We use the same derivation for :

==from here it is the old soln==

The solution to this quadratic equation is

Returning to the expression for the error in x, we have

We chose the signs to make the quantity as small as possible, making our allowed error minimal. This quantity is a constant: F and are chosen, and m, q, B, and xtot are properties of monopoles and of the field region. Many times in this derivation, we used constant, “typical” values of numbers which should truly depend on position. The field strength in the galactic region is somewhat homogeneous, and traversals must spend time in the most AND the least strong regions (not just one of the two), so to some degree the difference in strengths will balance out. To be safe, we account for this by taking smaller steps than what this formula suggests (see section \*^\*).

Now for the velocity done again.

The velocity is slightly more complex. (also maybe I’ll just use an upper bound for the number of steps as well, rather than an approximation based on how many have happened so far. Then I won’t have to recalculate every step… that makes a lot more sense actually. We’ll see how much I typically get with this formula first).

//I redid this one too because it makes sense to have vfinal be at most c.

We construct the final error in the x-velocity after a path is finished. For brevity we name the x-velocity simply . The final x-velocity in terms of the total number of steps and the average velocity stepsize is

The ratio of the final number of steps to the number of steps at some other time is approximately equal to the proportions of the final and current path lengths. In other words .

We use an over-estimate for to be safely within our error: for this we choose double the radius outside of which we stop tracking the particle, . \*footnote If the program finds its total length to be more than , we re-run the program with this approximated as 1.5 times the actual arc-length traversed.

We want the error in our final x-velocity to be less than 1%. The greatest our final x-velocity will be is the speed of light, so we require that

Substituting in equation \*^\*((vf, above)) :

Solve for :

To address all three directions, we require that all three of the steps in velocity be less than this quantity.

The total error is desired to be less than one percent:

Note that if L is the actual path length traversed,

Because we do not know the actual value of L before the algorithm is complete, we use a larger-than-expected value of the total path length in place of L, giving an upper bound:

Now, replacing this expression into equation (\*elephant, above under “the total error is…”) and solving for Δstep:

All other variables in equation (?? above for Δstep) are determined, except vfinal. We now find an upper bound on vfinal to complete the expression.

If a particle traverses a distance L through the magnetic field of force F=qB starting with v=0, we have by the work-energy theorem that

We note that the greatest change in velocity would be in a field with B pointing constantly in the same direction through the entire length , and with a large B for the milky way, BL. Using these variables in for L and B respectively and solving for v, we obtain

The final velocity vfinal will be greatest if the field constantly pushes in the direction of the initial v0, speeding the particle up. In this case, the initial speed and change in speed would add to the final speed:

Now,

Thus, to get less than 1% error, we require that the error per step in velocity be less than

In the adaptive-step 5th order Runge-Kutta, the error per step is approximated to be the difference between the 5th and 4th order steps.

**Notes**

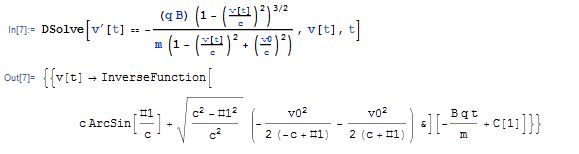
I. Future improvement: The biggest fault in the program is the time it takes to process. The time varies greatly with the initial velocity, because accelerations are small. Tracking the particle for a distance of order 1kpc would take days. To improve the program, I will continue seeking ways to better optimize the processing time. In addition, there may be a more accurate way to track the particle by making a higher-order approximation than just a constant velocity to calculate the acceleration. It is possible that such an approximation would take more time to calculate than just changing the timestep by an amount that adds the same accuracy, but it is still worth comparing the two.

II. Personal note: In retrospect, the theory doesn’t seem bad at all (other than somewhat lengthy expressions), but having been pretty unfamiliar with relativity (we learned about length contraction and time dilation in Physics I but that’s about it), I made a lot of errors and it took a lot more time than I had anticipated. I also spent a lot of time fixing bugs and making small adjustments.

**Questions**

1. On the line x=0,y=0 the magnetic field reduces to (BX is a parameter from the model). So the force is entirely in the z direction for a positive charge, and the equation for the z-acceleration reduces to (with vx, vy =0). Let the starting velocity in the z direction simply be and the z-acceleration be a.

Which has the solution:

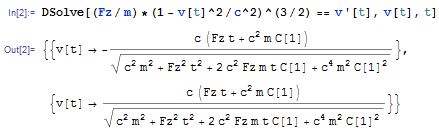


(not good!)

(incorrect solution: algebra error, below)

I had

Which has the solution:

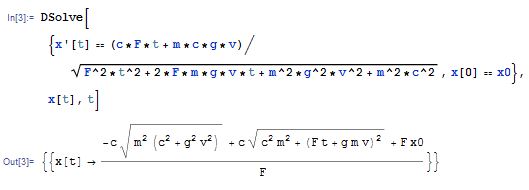


With the initial condition we get that

This yields

Simplifying this yields the condition that the from must be a +. We assume that a positive force in z must yield a positive velocity (if is positive as well), and from there we deduce that the in the differential equation solution must also be a +. Thus we have:

Solving this for x(t) with the condition x(0)=x0, we get



Sig figs reasoning:

in kpc/s, c9.715611713408621\*10-12 . To identify the number of sig-figs necessary to keep, we found the limiting velocity at which python considers (v/c)^2 to be 1, or γ=infinity. This happens soon after v=9.7156…0862 \*10-12. So we let c=9.7156…08621, a value of v at which γ=inf. Any more sig figs are redundant. (sum this bit up in a smaller amount). (question this – we are inaccurate in velocity so I don’t think we need that many)

-Low mass (1000(?)TeV) leads to almost immediate relativistic velocities, with accelerations of ~300m/s/s.

**Masses:**

>~ 10^4GeV: avoids standard model violations (wick 2002, pg3)

~10^5GeV: 3-family non-susy models

~10^7GeV: extra dimensions a la kaluza-klein, maybe(?) compact, like ~1mm leads to this mass.

~10^8GeV: exists in SU(15)

<10^11GeV: a comparison of the kibble flux to the parker limit (Wick 2003,pg666 RHS)

< 10^13GeV: from observations of curvature of universe(wick 2002, pg4 bottom)

<~ 10^15GeV: lack of proton decay mandates this

~10^17GeV: minimal SU(5) breaking

in short: , with some more likely spots in between

**Monopole Density**

From sloan’s paper on finding particles, a suggested density based on the kibble mechanism is “one monopole per (100km)­3