|  |
| --- |
| The Trashcan (Software and Media Publication, NFP) Presents |
| Applying Mathematics and Terrestrial Gravitational Motion in 2D Settings |
| A Physics Supplement for Application to Software, Including a Primer or Review of Analytical Geometry, Basic Calculus, and Newtonian Kinematics, Plus TerrestrialGravitation, a C++ Library |

|  |
| --- |
| Ray Arias  10 March 2018 to 29 April 2018 to Future Date |

Copyright © 2018 Raymond Arias, Jr. for The Trashcan Software and Media Publication. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License" and also available at http://www.gnu.org/licenses/ .

The accompanying software library, TerrestrialGravitation, is Copyright © 2018 Raymond Arias, Jr. for The Trashcan Software and Media Publication and is released under the conditions of the GNU General Public License. A copy of the license is included in the section entitled “GNU General Public License”.

TerrestrialGravitation is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

TerrestrialGravitation is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty or MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

A copy of the GNU General Public License is included at the end of this document in the section entitled “GNU General Public License” and also available at http://www.gnu.org/licenses/ .

**I humbly dedicate this supplemental paper and the accompanying software library to my loving wife, Maria A. Arias, for whose ceaseless work and assistance I will always be grateful.**

What is the first derivative of a cow? Prime rib!

—Numerous joke sites on the Internet

Falling in love is not at all the most stupid thing that people do, but gravitation cannot be held responsible for it.

—Albert Einstein

**Introduction**

You need to make a game, simulation, or other software in 2D and you need to account for the effect of gravity upon the objects in it? It’s not a problem. You just need to learn (or brush up on) your **terrestrial kinematics**. This is a basic Newtonian description (as opposed to Relativistic or Quantum and all that fancy newer stuff) of how gravity and motion work here on Earth’s surface (as opposed to in Earth’s center or in outer space or inside the sun, etc.).

As a note, there are other forces that can affect the path of physical objects in motion other than gravity. There is friction, both static and kinetic, with gaseous air particles, but it is very small for most objects, except when dealing with winds of great velocity, as in tornados and hurricanes, or taking into account objects that can be greatly affected by wind resistance or can take advantage of aerodynamic lift, such as gliders, airplanes, and parachutes. The physics of wind resistance and aerodynamic lift are extremely complicated and are not covered in this short supplemental paper.

**Alternate Forms of 2D Notation**

I would like to explain that there are a number of ways to notate the same 2-dimensional point on a Cartesian (rectangular) plane, both mathematically and geometrically. For example, the same point *P* can be notated as coordinates (*a*, *b*), or as a complex number *a* + *bi*, or as the geometric vector *a****î*** + *b****ĵ*** (***î*** and ***ĵ*** are unit vectors in the positive *x* and *y* directions respectively to express a geometric vector using components) [[1]](#footnote-1).

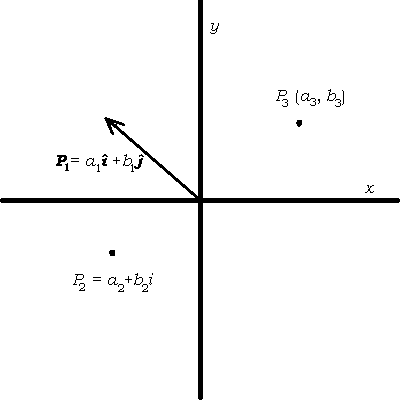
Comparison can only be done as to ascertain whether or not two points, such as *P*1 and *P*2, are equal or not, and *not* as to whether one is greater than or less than the other. Because each of the two points has two different values, neither one can be said to be greater than or less than the other. However, both points are equal, *if and only if*, both *a* values are equal *and* both *b* values are equal. These alternate systems of notation give rise to a number of operations that can be done to two or more points, such as finding equality, as well as addition and subtraction:

Figure 1: Example of points on a Cartesian plane, where *P*1 is a geometric vector, *P*2 is a complex number, and *P*3 is a pair of coordinates.

Although, operations as simple as addition, subtraction, and finding equality can be done using notation from any of these systems, others such as complex multiplication, complex division, and geometric vector dot product only have definitions specific to their notation system:

There is also a geometric vector cross product. However, since this cross product would introduce a third dimension and this paper only covers two-dimensional math, this will not be discussed in this paper.

**Distance from the Origin and Angle from the Right-Ascending Horizontal**

The distance from the origin to a point, which can also be construed as the absolute value of a complex number or the magnitude of a geometric vector, can be figured using the Pythagorean Theorem and adding the squares of the coordinates of the point, of the real and imaginary numbers in the complex number, or of the unit vector coefficients of the geometric vector, and then taking the square root of the result. In each equation below, the absolute value bars (**|***x***|**) indicate the following of the item between them: the distance, in the case of the point *P* with coordinates (*a*, *b*); the absolute value, in the case of the complex number *q* with real number component *a* and imaginary number component *b*; or the magnitude, in the case of the geometric vector with ***î*** unit vector coefficient *a* and ***ĵ*** unit vector coefficient *b*.

Although, comparatively speaking, calculating the distance/absolute value/magnitude of a point/complex number/geometric vector is relatively simple, determining its angle from the horizontal right semiaxis, the positive *x*-axis, is a bit more complicated. This is due to the angular values applicable to every inverse trigonometric function (also known as an “arc function”) occurring every ½ rotation of the full circle. This, in turn, is due to the given values yielded by each standard trigonometric function repeating its values cyclically. Therefore, for every possible resulting value of any given standard trigonometric function, there are two possible angular values in the full 2π-radian, or 360°, rotation that correspond to this original value.

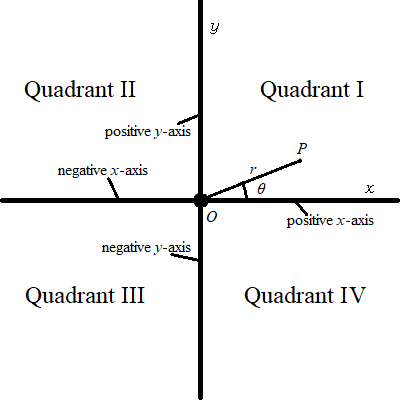
Nonetheless, because we already know the coordinates/real and imaginary components/unit vector coefficients, we can know what quadrant it is in (or which axis it is on, if applicable, or if it is the origin). Using this information in conjunction with the result of an inverse trigonometric function, such as the arctangent, an exact angle on the rotational circle can be calculated.

Figure 2: Cartesian plane showing the distance *r* from the origin *O* to a point *P*, the angle *θ* the distance line makes with the horizontal, the four Quadrants, and the negative and positive parts of both axes.

For the sake of simplicity, let us consider a point *P* with coordinates (*a*, *b*). If *P* is in Quadrant I (also known as the First Quadrant), above the *x*-axis and to the right of the *y*-axis, both *a* and *b* would be positive. But if *P* is in Quadrant II (also known as the Second Quadrant), it would still be above the *x*-axis, but this time to the left of the *y*-axis, *a* would be negative and *b* would be positive. Further, if *P* is in Quadrant III (also known as the Third Quadrant), it would now be both below the x-axis **and** to the left of the *y*-axis, and both *a* and *b* would be negative. Finally, if *P* is in Quadrant IV (also known as the Fourth Quadrant), it still be below the *x*-axis, but now to the right of the *y*-axis, and therefore, *a* would now be positive, but *b* would still be negative.

An integer function can be created using this information that gives 1 if P is in the First Quadrant, 2 if P is in the Second, 3 if it is in the Third, or 4 if in the Fourth. However, there are a few places on the Cartesian plane that don’t fall in any of the four Quadrants. These locations are along the x-axis, along the y-axis, and on the origin (which can be considered as being along both axes or neither). If the function we wish to create, which we can mathematically define as *quadrant*(*a* ϵ ℝ, *b* ϵ ℝ) ϵ ℤ | 1 ≤ *quadrant* ≤ 4 (or, “quadrant is a function of a, member of real numbers, and b, member of real numbers, which function value is a member of integers, such that quadrant is equal to or between 1 and 4”), we can define the *quadrant*(*a*, *b*) function to be *quadrant*(0, 0) = 0 when *P* is the origin, and then give this function negative values when *P* lies somewhere else on one of the axes. Let’s allow *P* on the positive *x*-axis yield *quadrant*(*a*, 0) = –1 (*a* > 0), on the positive *y*-axis give *quadrant*(0, *b*) = –2 (*b* > 0), on the negative *x*-axis make for *quadrant*(*a*, 0) = –3 (*a* < 0), and on the negative *y*-axis result in *quadrant*(0, *b*) = –4 (*b* < 0). We must now expand the range of the *quadrant*(*a*, *b*) function to be –4 ≤ *quadrant* ≤ 4 (or, *quadrant* is equal to or between -4 and 4). We can now create a complete table of every possible value for our *quadrant*(*a*, *b*) function as follows:

|  |  |
| --- | --- |
| **Location of point *P*** | **Value of *quadrant*(*a*, *b*)** |
| Negative *y*-axis (*a* = 0, *b* < 0) | –4 |
| Negative *x*-axis (*a* < 0, *b* = 0) | –3 |
| Positive *y*-axis (*a* = 0, *b* > 0) | –2 |
| Positive *x*-axis (*a* > 0, *b* = 0) | –1 |
| Origin (*a* = 0, *b* = 0) | 0 |
| Quadrant I (*a* > 0, *b* > 0) | 1 |
| Quardant II (*a* < 0, *b* > 0) | 2 |
| Quadrant III (*a* < 0, *b* < 0) | 3 |
| Quadrant IV (*a* > 0, *b* < 0) | 4 |

As mentioned above, once we can identify the Quadrant *P* is in, we can use this information to know which of the two angles compatible with the arctangent function is the correct one. Basically, we obtain the slope of the distance line by dividing *b* by *a*. Next, we use this slope-quotient and put it into the arctangent function, which customarily yields an angle (in radians) -π/2 < *φ* < π/2. If we require the resultant angle to be 0 ≤ *θ* < 2π, then when *quadrant*(*a*, *b*) = 4, we can add 2π to the angle yielded by the arctangent function. Since a point in Quadrant II would yield a negative slope-quotient *b*/*a* just like one in Quadrant IV, when *quadrant*(*a*, *b*) yields 2, we need to add π to this angle to obtain the correct one, and since a point in Quadrant III gives a positive slope-quotient, just like one in Quadrant I, when *quadrant*(*a*, *b*) yields 3, we also need to add π to the angle as well. Finally, for 0 and all the negative values yielded for *quadrant*(a, b), the appropriate angle is already known. For *P* at the origin (*quadrant*(0, 0) = 0), there is no valid angle as the distance is also 0. However, where a value is required, as in a software function, 0 would do just as good as any other. As for *P* on the positive *x*-axis, *θ* = 0; for *P* on the positive *y*-axis, *θ* = π/2; for *P* on the negative *x*-axis, *θ* = π; and for *P* on the negative *y*-axis, *θ* = 3π/2. The following is a mathematical summary of how the final angle is obtained, with the right arrow symbol (→) indicating the assignment to take place after it if the condition before it is met (for example A→B is read, “if A, then do B”), *φ* being the angle yielded by the arctangent function, and *θ* being the final desired angle.

Here are four examples of points from which we will obtain a distance from the origin and an angle from the right horizontal.

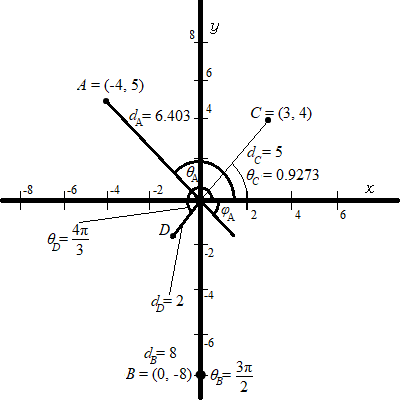


Figure 3: Plane showing example points, angles, and distances, including *φA* = –0.8961, *θA* = 2.246, and point *D* = (–1, ).

We now have a distance and an angle, which we can use to implement an entirely different coordinate system known as the polar coordinate system, whose coordinates are based on a radius, *r*, and an angle, *θ*, or, (*r*, *θ*). Further, these polar coordinates can be converted back to rectangular (Cartesian) coordinates using two much simpler calculations utilizing the sine and cosine functions: *x* = *r* cos *θ* and *y* = *r* sin *θ*. For example, in the last example above we have *d* = 2 (we’ll just use this *d* as our *r*) and *θ* = 4π/3, or the point *D* = (2, 4π/3). Plug these into our polar-to-rectangular formulas and we have *x* = 2 cos 4π/3 and *y* = 2 sin 4π/3. This yields *x* = 2(–1/2), *x* = –1 and *y* = 2(/2), *y* = , or *D* = (–1, ), which is what we started out with.

**Some Basic Calculus**

Before we go on, let us cover some other basics. If you don’t know calculus, that’s fine, but you will have to understand a few of its most essential concepts. Given a function *y* = *f*(*x*) (“f of x”), its **derivative** (“d y d x equals f prime of x”) is the rate of change of the function (or *y*) with respect to its parameter value (*x*), or more fundamentally, the slope of the function at the point (*x*, *y*). It should be noted here that the differentials (*dy* and *dx*) written as a fraction do in no way signify any actual rational fraction, it’s just one of a number of convenient ways to notate the derivative of a function. The derivative of a derivative is known as the **second derivative** of a function and is notated (“d squared y d x squared equals f double prime of x” or “…f second of x”). Further, the area between the *y*-axis and the curve made by a function, adding when the curve is above the *y*-axis and subtracting when the curve is below it, from 0 to *x* for *x* > 0, or *x* to 0 for *x* < 0 gives you its **integral**. Taking the integral of a derivative of a function will give you back the original function, particularly if you know any constants (such as C) that were part of the original function that got obliterated by the derivative. Here are four simple examples of how this occurs, the last of which shows a second derivative and its reintegration:

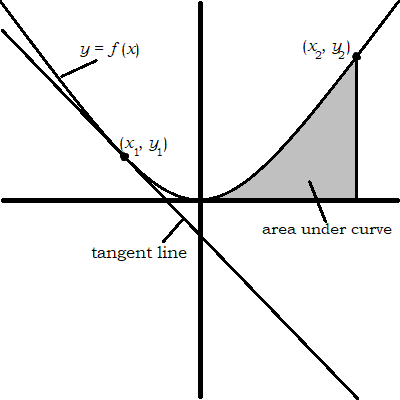


Figure 4: The slope of the tangent line gives the derivative of *f* (*x*) at (*x*1, *y*1) and the area under the curve gives the integral of *f* (*x*) at  
(*x*2, *y*2).

Notice that the derivative of any constant is always zero. (*f*(*x*) = *a*, *f’*(*x*) = 0) Also, the derivative of a linear function of *x* is just the coefficient of *x* as a constant. (*f*(*x*) = *ax*, *f’*(*x*) = *a*) Conversely, the integral of any constant is this constant multiplied by *x* (plus another constant C). Further, when taking the derivative of a quadratic function, the coefficient of *x*2 is multiplied by 2 and *x*2 becomes *x*. (*f*(*x*) = *ax*2, *f’*(*x*) = 2*ax*) Also, just as before the coefficient of *x* stands alone as a constant and is added to the previous term. (*f*(*x*) = *ax*2 + *bx*, *f’*(*x*) = 2*ax* + *b*) Reversing this process, in integrating a term with *x*, the coefficient is divided by 2 and *x* becomes *x*2. Just in **differentiation** (taking the derivative), terms also add together during **integration**.

**The Calculus of Kinematics**

When examining kinematics, we do not observe *y* with respect to *x*, by rather *x* and *y* each with respect to time (*t*). So an object’s position is actually *x*(*t*) and *y*(*t*), but we generally still use the conventional notation *x* and *y*. But the time parameter becomes apparent when we take the position’s derivative to obtain velocity, such as . The dot on the x (pronounced “x dot” or “x dot of t”) is the same as the prime sign above, except the derivative is taken with respect to *t*, as opposed to another variable.

The **position** (*x*, for horizontal position or *y*, for vertical position) of a physical object is its placement in relation to other objects or a coordinate system, such as the Cartesian plane. When the object is in motion, its **velocity** (*vx*, for horizontal velocity or *vy*, for vertical velocity) is the rate of change in its position with respect to time (*t*). This rate of change is expressed as either of the derivatives or . When this object’s velocity changes, the rate of this change is its **acceleration** (*ax* or *ay*), also known as or .

Just as the rates of change can be arrived at by taking derivatives, given a rate of change, the original kinematic attribute can be arrived at by taking an integral. So taking the integral of the acceleration of an object will give you back its velocity, assuming you know its initial velocity, and taking the integral of the velocity will give you back its position, assuming you know its initial position.

Terrestrial kinematics entails **horizontal kinematics**, which as you may guess is a description of horizontal motion, and **vertical kinematics**, a description of vertical motion. Here is each one in detail.

**Horizontal Kinematics**

Since any forces pushing or pulling upon any object in the horizontal while it is flying through the air are negligible, a solid object being thrown, catapulted, or otherwise projected into the air, can generally be considered as having neither any thrust added to, nor any resistance taken away from, its initial motion. Hence, the horizontal speed is practically constant; there is no horizontal component of acceleration to take into account. Consequently, the horizontal position only changes by adding the same constant as a coefficient of time to the initial position when animating in 2D software.

Notice how the initial kinematic attributes (*x*(0) and *vx*(0)) are constants and act the same way C does in integrals of functions of *y* with respect to *x*, like those in the introduction.

**Vertical Kinematics**

The vertical motion of objects flying through the air, for instance after being catapulted or thrown, is similar to that of an object, such as a pinball,, loaded on top of a coiled spring and then propelled by allowing the spring to uncoil. This object would then be shot upward by the force of the uncoiling spring. This upward force would immediately be counteracted by the downward force of terrestrial gravity. Once the object is out of contact with the spring, the object would continue upward with whatever velocity was given to it initially by the force of the spring. This upward velocity would reduce until it reaches zero as gravitation would be the only force affecting the object after that point. Additionally, after reaching its vertex (the highest point in its vertical path), the object would then gain a downward velocity from the continued gravitation affecting it. This velocity would be small at first and then grow to its maximum just before the object either reloads the spring, recoiling it in the process, or it hits the ground. Generally, a vertically propelled object returns with a downward velocity of the same magnitude, at the same height that it started, as the initial upward velocity that it was originally driven to by the upward force. For example, if a ball was thrown straight up at 10 miles/hour, by the time it finishes going up, gets to its vertex, and goes down, when it reaches the same height it was thrown up at, it should be going straight down at pretty much (if not *exactly*) 10 mi/hr.

After this point, if an object with downward velocity is allowed to make contact with the ground, depending upon what the ground is made of and what the object is made of (as well as the shape of the object and which part makes contact with the ground), the object will bounce back up at some fractional upward velocity of the downward velocity it came down with. For a rubber ball on a hard surface, this fraction is very large, near one, for a less bouncy object on a plush surface, this fraction is very small, near zero. This fraction is known as the coefficient of restitution (CoR, or *epsilon* ε). Each subsequent bounce results in CoR being reapplied until the y component of velocity of the object is close enough to zero as to be negligible.

In physics, a **force** gives rise to the acceleration of an object. A force is calculated using Isaac Newton’s equation *F* = *ma*, where *m* is the mass of the object and *a* is its acceleration. Terrestrial gravitation is one such force and the equation for it is a special case of Newton’s general force equation. The terrestrial force of gravity upon any object’s mass is *G*⊕,which is equal to *mg*, the mass of the object (*m*) multiplied by the acceleration given to it by the gravitation between the object and the Earth (*g*). This acceleration is expressed either in US Customary Units as 32.1740 feet/sec2 or in the International System of Units (SI) as 9.80665 meters/sec2. These are averages for mid-latitudes at average altitude and actual measurements for *g* can vary according to altitude from sea level as well as latitude on the Earth’s surface.

If these facts are applied separately to each of the axes, when plotted on a graph using sample initial velocities for each direction, the following is obtained:

If both axes are applied jointly, this graph is obtained:

**Conclusion**

You may be wondering what all this math and physics means for creating games, simulations, and other software. Well, I understand some people reading this may be having a hard time with math formulas, particularly those involving calculus, but basically, here is the basic breakdown. Since, on the horizontal axis, the velocity is going to be constant, you can assign a constant velocity

const double vx = 10.0;

x = x0 + (vx \* t);

(where vx is horizontal velocity, x is current horizontal position, x0 is initial horizontal position, and t is time) or neglect it entirely as you can just hard code the constant directly when you are changing the x-coordinate (x = x0 + (10 \* t);). For the vertical velocity, you will need a variable though. It is your choice, however, if you want to keep a constant, such as,

const double g = 9.8;

vy = vy0 – (g \* t);

y = y0 + (vy \* t);

(where g is the gravitational acceleration constant, vy is the current vertical velocity, vy0 is the initial vertical velocity, y is the current vertical position, and y0 is the initial vertical position) or hard code when you change your vertical velocity.

vy = vy0 – (9.8 \* t);

y = y0 + (vy \* t);

Nevertheless, keep in mind that 9.8 (m/s)/s—or whichever number you select for *g*—is a change that occurs every whole second, so if you are changing your vertical velocity 100 times per second, you will need to divide this number by 100. Thus, you should use either something like

const double g = 9.8;

vy = vy0 – ((g / 100) \* (t / 100);

or,

const double g100 = 0.98;

t100 = t / 100;

vy = vy0 – (g100 \* t100);

or even,

const double g = 9.8;

dt = 100;

vy = vy0 – ((g / dt) \* (t / dt));

Whichever works for you, when your object finally flies across the screen, you should get a similar shape to the second graph (concave down parabola). If not, you need to double check your math. Good luck!

**Appendix: Reference for the C++ Library TerrestrialGravitation**

I hope you find bundled with this document a software library I made in C++. This is to help you create objects that behave as if they were under the influence of terrestrial gravitation as well as in order to instruct you as to how to code such objects and the behavior underlying them appropriately. I call it, simply enough, TerrestrialGravitation.

By the way, I have decided that anyone who wants to use this commercially and proprietarily can either afford other software that would effectively do what this library already does, or can afford to effectively recode it from scratch by reverse engineering it, which they are more than welcome to do. This is why this library is released under harder CopyLeft license of GNU GPL, v3, rather than the lesser so LGPL.

These classes have constructors as fully defined as possible, from default constructors with no parameters, to fully defined constructors with every variable member defined as a parameter, to copy constructors, to assignment operators, to even destructors.

**class Cartesian**

The library starts with the class Cartesian. These are just objects that have integer x and y coordinates (int), as well as double precision floating point coordinates (double), designated as xx and yy. I made two sets of coordinates so that switching between doubles and ints would not be any excessive burden. For scientific precision, doubles are best used. However, for games and graphic simulations that require pixel coordinates, ints are much faster and convenient to use. Therefore, the Cartesian class has coordinates of both types.

Due to this, I inserted three functions to ease using objects that come out of this class. First is the Boolean function bool concordance(), which requires no parameters. This checks to see if the values of x and xx are within less than an integer unit’s difference equal. If this test passes, it checks for the same level of equality between y and yy. If this test passes, TRUE is returned, otherwise the result is FALSE. Next, is the void function concordanceInt(). Basically, all this does is force concordance, or agreement, between the ints and the doubles. In the case of concordanceInt(), this is accomplished by assigning the doubles to the values of the ints (casting them as doubles in the process, of course). Finally, the void function concordanceDouble() assigns the values of the doubles (casted as ints, and therefore, truncated) to the ints.

Notwithstanding the assignment operator (operator=(…)), there is also an equality operator (operator==(…)) and an in equality operator (operator!=(…)) that will evaluate to true and false respectively if two Cartesian objects are equal and unequal, and *vice versa* if not. I have also placed an addition operator (operator+(…)), as well as a subtraction operator (operator-(…)), in order to facilitate the addition and subtraction of coordinates of different Cartesian objects. There are also a multiplication operator (operator\*(…)), as well as a division operator (operator/(…)), in order to carry out complex multiplication and complex division. These operators treat the *x*-coordinates of both Cartesian objects as the real parts of two complex numbers and the *y*-coordinates as the imaginary parts and calculate the complex product or complex quotient according to the following formulas (where *c*1 is the first Cartesian object, *c*2 is the second Cartesian object, *p* is the complex product, and *q* is the complex quotient):

There are functions for finding distance from the Cartesian object to the origin, or between two Cartesian objects. This can be given as a double or an int, according to if you use distanceInt(…) or distanceDouble(…). Computing distance is done by using the Pythagorean theorem: *a*² + *b*² = *c*², or in this case, , where ***distance*** ≥ 0.

There are a trio of functions, quadrant(), quadrant(int w, int z), and quadrant(double ww, double zz), that each return an int for the Quadrant of the Cartesian object or the point given by (w, z) or (ww, zz). The values returned are 1 for Quadrant I (*x* positive, *y* positive), 2 for Quadrant II (*x* negative, *y* positive), 3 for Quadrant III (*x* negative, *y* negative), 4 for Quadrant IV (*x* positive, *y* negative), -1 for positive *x*-axis, -2 for positive *y*-axis, -3 for negative *x*-axis, -4 for negative *y*-axis, or 0 for the origin (0, 0).

There are also functions that return the angle pointing away from the origin or the other Cartesian object. The angleInt(…)functions return degrees and the angleDouble(…)ones return radians. However, if you use angleInt(…), the doubles will be concordanceInt()-ed with the ints, and if you use angleDouble(…), vice versa. Angles are calculated using the arctangent of the slope of the distance line, or , or in C code: double thetaRad = atan(yy / xx); the -Int version carries out this same calculation but multiplies the result by the constant 180/π, or again in C: int thetaDeg = (int) round((180.0 / PI) \* thetaRad); However, due to the limits of the arctan function, we must find the quadrant or axis the object is on, as you will see. Here are the formulas, where *q* is the integer result of the quadrant(…) function, *φ* is a working angle and *θ* is the final angle returned (A → B, in this context, means if A is true then do B):

The arctan function only normally gives back an angle of . And to further complicate things, as the angle gets closer to –π/2 or π/2, the closer gets to negative or positive infinity. Therefore, using arctan alone to find an angle is not very effective without first knowing which quadrant the point is in or which axis it is on. So these functions make use of the quadrant(…) functions above. If the point is on the positive *x*-axis or in Quadrant I, the raw arctan result is returned. If the point is on the positive *y*-axis, π/2, or 90°, is returned. If the point is in Quadrant II, π, or 180°, is added to the angle. If the point is on the negative *x*-axis, π, or 180°, is returned. If the point is in Quadrant III, π, or 180°, is added to the angle. If the point is on the negative *y*-axis, 3π/2, or 270°, is returned. Finally, if the point is in Quadrant IV, in order for all angles returned to be positive in the range of , or , 2π, or 360°, is added to the angle. By the way, if the point is on the Origin, a default angle of 0, or 0°, is returned, however, it should be noted that such a point would have no valid angle.

There are some functions that return the *x*-coordinate or *y*-coordinate of a given distance and angle pair (also known as polar coordinates (*r*, *θ*)) (*r* is the distance, or radius, and *θ* is the angle from the horizontal right semiaxis, also known as the positive *x*-axis) as an int (rectangularX(int r, int thetaDeg) and rectangularY(int r, int thetaDeg)) or as a double (rectangularXX(double rr, double thetaRad) and rectangularYY(double rr, double thetaRad)). Further, there is a pair of functions that return a Cartesian object from an (*r*, *θ*) pair given as ints (*θ* in degrees) (rectangularC(int r, int thetaDeg)) or as doubles (*θ* in radians) (rectangularC(double rr, double thetaRad)) as well as another pair of functions that does the same thing, except, instead of returning a Cartesian object, the values are stored in the current Cartesian object. This is accomplished using the trigonometric functions sine and cosine for finding the y-coordinate and the x-coordinate of the point respectively. Namely, *x* = *r* cos *θ* and *y* = *r* sin *θ*.

There are functions for reading and assigning either or both of the ints x and/or y or either or both of the doubles xx and/or yy. After each assignment, one of the concordance assignment functions is executed so that agreement is maintained between ints and doubles.

There are also functions for inverting (giving the negative of) ints x and/or y or doubles xx and/or yy as well as any int or double passed as a parameter. This may be useful in certain contexts, such as drawing graphics where most raster screens put the Origin (0, 0) at the top left and the x-coordinate increases to the right while the y-coordinate increases downward.

There is an atOrigin() function that assigns everything to zero (referred to on Cartesian graphs as the Origin, or (0, 0)). In order to facilitate assignment to another Cartesian object, atOrigin() returns the result as well as assigning it to the current object (\*this).

Lastly, there is a function that will return the geometric vector dot product of the current Cartesian object and another Cartesian object given as a parameter. It does this by computing the following formula, where carteA is the current object and carteB is the object given as a parameter:

The class Cartesian, as you will see, is a fully accessible encapsulated class of class InertBody. All members are public, so they are all freely accessible. However, two Cartesian objects (position and velocity) will be used in class InertBody. Also, all members of class Cartesian are accessible by way of functions (for instance, int X() and void X(int a) to read and assign the Cartesian member int x respectively, etc.). There are also functions to write members jointly (void XY(int a, int b) and void XXYY(double aa, double bb)).

*Variable Members:* ints: x, y, doubles: xx, yy.

*Constructors:* Cartesian(void): (***default constructor***) assigns 0 to x and y and 0.0 to xx and yy.

Cartesian(int a, b): assigns a to x and b to y and executes concordanceInt() to make the doubles agree with the ints.

Cartesian(double aa, bb): assigns aa to xx and bb to yy and executes concordanceDouble() to make the ints agree with the doubles.

Cartesian(const Cartesian &cartecopy): (***copy constructor***) takes each member of a given Cartesian object and assigns its value to each member of the current object.

*Operators:* ***Assignment*** (Cartesian operator=(const Cartesian &carteassignment)): Same as ***copy constructor***, but checks for self-assignment, for which case nothing happens, and at the end the current object (\*this) is returned.

***Negation*** (Cartesian operator-(const Cartesian &cartenegative)): Returns the negative of a Cartesian object by assigning double –cartenegative.xx to double xx, assigning double –cartenegative.yy to double yy, and executing concordanceDouble().

***Addition*** (Cartesian operator+(const Cartesian &carte1, const Cartesian &carte2)): Adds the coordinates of two Cartesian objects and returns a third whose coordinates are the sum of the other two.

***Subtraction*** (Cartesian operator–(const Cartesian &carte1, const Cartesian &carte2)): Subtracts the coordinates of a second Cartesian object from the coordinates of the first one and returns a third object whose coordinates are the difference of the other two.

***Multiplication*** (Cartesian operator\*(const Cartesian &carte1, const Cartesian &carte2)): (***complex product***) Returns the complex product of two Cartesian objects by assigning (double carte1.xx \* double carte2.xx) – (double carte1.yy \* double carte2.yy) to double xx, assigning (double carte1.xx \* double carte2.yy) + (double carte1.yy \* double carte2.xx) to double yy, and executing concordanceDouble().

***Division*** (Cartesian operator/(const Cartesian &carte1, const Cartesian &carte2)): (***complex quotient***) Returns the complex quotient of two Cartesian objects by creating doubles a, b, c, d, qr, and qi, and making the following assignments:

a = carte1.xx; b = carte1.yy;

c = carte2.xx; d = carte2.yy;

qr = ((a \* c) + (b \* d)) / ((c \* c) + (d \* d));

qi = ((b \* c) – (a \* d)) / ((c \* c) + (d \* d));

Then qr is assigned to double xx, qi is assigned to double yy, and finally concordanceDouble() is executed.

***Equality***(bool operator==(const Cartesian &carteEqual1, const Cartesian &carteEqual2)): assigns false to a new bool equality and tests carteEqual1.concordance() and carteEqual2.concordance(). If both are true, double carteEqual1.xx is compared to double carteEqual2.xx to see if they are equal. If this result is true, double carteEqual1.yy is compared to double carteEqual2.yy to see if they are equal. If this result is true, true is assigned to equality. If any of the above tests fail, false will remain assigned to equality. Finally, equality is returned.

***Inequality***(bool operator!=(const Cartesian &carteUnequal1, const Cartesian &carteUnequal2)): assigns false to a new bool inequality and tests carteUnequal1.concordance() and then carteUnequal2.concordance(). If either is false, true is assigned to inequality. If both of these are true, double carteUnequal1.xx is compared to double carteUnequal2.xx to see if they are unequal. If so, true is assigned to inequality. Otherwise, double carteUnequal1.yy is compared to double carteUnequal2.yy to see if they are unequal. If so, true is assigned to inequality. Otherwise, if all these tests fail, false remains assigned to inequality. Finally, inequality is returned.

*Destructor:* ~Cartesian(void): deletes current Cartesian object.

*Functions for reading the current position:*

int X(void): returns the current object’s value for int x.

int Y(void): returns the current object’s value for int y.

double XX(void): returns the current object’s value for double xx.

double YY(void): returns the current object’s value for double yy.

Cartesian Position(void): returns a Cartesian object whose coordinates are identical to the current object’s coordinates.

*Functions for setting the current position:*

void X(int a): assigns a to int x, executes concordanceInt().

void Y(int b): assigns b to int y, executes concordanceInt().

void XY(int a, int b): assigns a to int x, assigns b to int y, executes concordanceInt().

void XX(double aa): assigns aa to double xx, executes concordanceDouble().

void YY(double bb): assigns bb to double yy, executes concordanceDouble().

void XXYY(double aa, double bb): assigns aa to double xx, assigns bb to double yy, executes concordanceDouble().

void Position(Cartesian pos): assigns pos’s coordinates to current object’s coordinates by way of assigning pos.xx to xx, assigning pos.yy to yy, and executing concordanceDouble().

*Functions that invert (negate) coordinates or numbers:*

void invertX(void): transforms x and xx to –x and –xx.

void invertY(void): transforms y and yy to –y and –yy.

void invertXY(void): transforms x, xx, y, and yy to –x, –xx, –y, and –yy.

int invert(int a): returns –a.

double invert(double aa): returns –aa.

*Functions that compute distance*:

int distanceInt(void): calculates distance from Cartesian object to origin (0, 0) to nearest integer unit and returns result.

double distanceDouble(void): calculates distance from Cartesian object to origin (0, 0) and returns result.

double distance(void): same as distanceDouble().

int distanceInt(int a, int b): calculates distance between from current Cartesian object’s coordinate to coordinates given by (a, b) to the nearest integer unit and returns the result.

int distance(int a, int b): same as distanceInt(int a, int b).

double distanceDouble(double aa double bb): calculates distance from current Cartesian object’s coordinates to coordinates given by (aa, bb) and returns the result.

double distance(double aa, double bb): same as distanceDouble(double aa, double bb).

int distanceInt(Cartesian carte): calculates the distance between the current Cartesian object and another Cartesian object given by carte to the nearest integer unit and returns the result.

double distanceDouble(Cartesian carte): calculates the distance between the current Cartesian object and another Cartesian object given by carte and returns the result.

double distance(Cartesian carte): same as distanceDouble(Cartesian carte).

*Functions for finding angles:*

int angleInt(void): computes the angle from the current Cartesian object to the origin (0, 0) in degrees to the nearest integer unit and returns the result.

double angleDouble(void): computes the angle from the current Cartesian object to the origin (0, 0) in radians and returns the result.

double angle(void): same as angleDouble(void).

int angleInt(int w, int z): computes the angle in degrees from the current Cartesian object to coordinates given by (w, z) to the nearest integer unit and returns the result.

int angle(int w, int z): same as angleInt(int w, int z).

double angleDouble(double ww, double zz): computes the angle in radians from the current Cartesian object to coordinates given by (ww, zz) and returns the result.

double angle(double ww, double zz): same as angleDouble(double ww, double zz).

int angleInt(Cartesian carte): calculates angle in degrees from current Cartesian object to another Cartesian object given by carte to the nearest integer unit and returns the result.

double angleDouble(Cartesian carte): calculates angle in radians from current Cartesian object to another Cartesian object given by carte and returns the result.

double angle(Cartesian carte): same as angleDouble(Cartesian carte).

*Functions for finding the quadrant point is in or axis point is on:*

int quadrant(void): returns an int -4 ≤ q ≤ 4 indicating where the Cartesian object is located. Value returned according to this table:

|  |  |
| --- | --- |
| **Location of Cartesian object** | **Value Returned** |
| Negative *y*-axis (*x* = 0, *y* < 0) | -4 |
| Negative *x*-axis (*x* < 0, *y* = 0) | -3 |
| Positive *y*-axis (*x* = 0, *y* > 0) | -2 |
| Positive *x*-axis (*x* > 0, *y* = 0) | -1 |
| Origin (*x* = 0, *y* = 0) | 0 |
| Quadrant I (*x* > 0, *y* > 0) | 1 |
| Quardant II (*x* < 0, *y* > 0) | 2 |
| Quadrant III (*x* < 0, *y* < 0) | 3 |
| Quadrant IV (*x* > 0, *y* < 0) | 4 |

int quadrant(double ww, double zz): same as quadrant(), except returns q according to location of (ww, zz) instead of location of object.

*Functions for obtaining rectangular coordinates from a radius and angle:*

int rectangularX(int r, int thetaDeg): assigns r casted as a double to double rr, assigns thetaDeg (casted as a double) multiplied by the equivalent of π/180.0 to double theta,—converting thetaDeg from an int in units of degrees to a double in units of radians and assigning this value to theta—assigns rr times cos(theta) to double ww, and converts ww to an int by assigning round(ww) (which just rounds ww to the nearest integer value) casted as an int to int w before returning w.

double rectangularXX(double rr, double thetaRad): assigns rr multiplied by cos(thetaRad) to double ww and returns ww.

int rectangularY(int r, int thetaDeg): assigns r casted as a double to double rr, assigns thetaDeg (casted as a double) multiplied by the equivalent of π/180.0 to double theta,—converting thetaDeg from an int in units of degrees to a double in units of radians and assigning this value to theta—assigns rr times sin(theta) to double zz, and converts zz to an int by assigning round(zz) (which just rounds zz to the nearest integer value) casted as an int to int z before returning z.

double rectangularYY(double rr, double thetaRad): assigns rr multiplied by sin(thetaRad) to double zz and returns zz.

Cartesian rectangularC(int r, int thetaDeg): casts r to double and assigns it to double rr, casts thetaDeg to double, multiplies it by the equivalent of π/180.0, and assigns the result to double theta; assigns rr \* cos(theta) to double ww and rr \* sin(theta) to double zz; creates the Cartesian object point, assigns ww to double point.xx and zz to double point.yy; executes the function point.concordanceDouble(); and returns point.

Cartesian rectangularC(double rr, double thetaRad): assigns rr \* cos(theta) to double ww and rr \* sin(theta) to double zz; creates the Cartesian object point, assigns ww to double point.xx and zz to double point.yy; executes the function point.concordanceDouble(); and returns point.

void rectangular(int r, int thetaDeg): same as Cartesian rectangularC(int r, int thetaDeg) but uses the current Cartesian object to place the result in instead of returning another Cartesian object. It assigns double ww, which is calculated the same way, to double xx of the current object and double zz to yy before finally executing the function concordanceDouble() of the current object.

void rectangular(double rr, double thetaRad): same as Cartesian rectangularC(double rr, double thetaRad) but uses the current Cartesian object to place the result in instead of returning another Cartesian object. It assigns double ww, which is calculated the same way, to double xx of the current object and double zz to yy before finally executing the function concordanceDouble() of the current object.

*Functions for checking and forcing concordance of* int *and* double *coordinate sets:*

bool concordance(void): checks to see if (x, y) and ((int) round(xx), (int) round(yy)) agree to the nearest integer unit. Returns TRUE if they do and FALSE if they don’t.

void concordanceInt(void): forces concordance, or agreement, between double and int coordinates, doubles are assigned double casted values of ints.

void concordanceDouble(void): forces concordance, or agreement, between double and int coordinates, ints assigned int casted values of doubles after the double round(double) function from <cmath> is executed.

*Miscellaneous Function:*

Cartesian atOrigin(void): current Cartesian object is assigned to be at the origin (0, 0) and (0.0, 0.0) and is returned to facilitate assignment.

double DotProduct(const Cartesian &carte): Returns the geometric dot product of current Cartesian object and carte by creating doubles mag1, mag2, ang, and dot and making the following assignments:

mag1 = distanceDouble();

mag2 = carte.distanceDouble();

ang = angleDouble(carte);

dot = mag1 \* mag2 \* cos(ang);

Finally, dot is returned.

**class InertBody**

The class InertBody is an encapsulator class of class Cartesian in that class InertBody makes use of class Cartesian by creating two objects from it, Cartesian position and Cartesian velocity, that are members of this class. There is also a protected member, bool \_at\_rest, that every function that sets Cartesian velocity tests to be (0, 0) by way of declaring a new Cartesian Zero and then executing its atOrigin() function and then comparing velocity to Zero. This class only applies position and velocity, but neither any gravitation, nor any other force, hence the name of this class.

*Constant defined by precompiler:* \_DEFAULT\_TIME is 30.

*Variable Members:* Cartesian objects: position, velocity, initialPosition.

**Protected** *Variable Members:* bool \_at\_rest, ints: \_n, \_t.

*Constructors:* InertBody(void): (***default constructor***) executes Cartesian function atOrigin() for position, velocity, and initialPosition, sets bool \_at\_rest to TRUE, and assigns 0 to int \_n and \_DEFAULT\_TIME to int \_t.

InertBody(int a, int b): assigns (a, b) to Cartesian position, executes position.concordanceInt(), executes velocity.atOrigin(), executes Position() (sets initialPosition = position), sets bool \_at\_rest to TRUE, and assigns 0 to int \_n and \_DEFAULT\_TIME to int \_t.

InertBody(double aa, double bb): assigns (aa, bb) to Cartesian position, executes position.concordanceDouble(), executes velocity.atOrigin(), executes Position(), sets bool \_at\_rest to TRUE, and assigns 0 to int \_n and \_DEFAULT\_TIME to int \_t.

InertBody(Cartesian pos): assigns (pos.xx, pos.yy) to Cartesian position, executes position.concordanceDouble(), executes velocity.atOrigin(), executes Position(), sets bool \_at\_rest to TRUE, and assigns 0 to int \_n and \_DEFAULT\_TIME to int \_t.

InertBody(int a, int b, int time): assigns (a, b) to Cartesian position, executes position.concordanceInt(), executes velocity.atOrigin(), executes Position(), sets bool \_at\_rest to TRUE, and assigns 0 to int \_n and time to int \_t.

InertBody(double aa, double bb, int time): assigns (aa, bb) to Cartesian position, executes position.concordanceDouble(), executes velocity.atOrigin(), executes Position(), sets bool \_at\_rest to TRUE, and assigns 0 to int \_n and time to int \_t.

InertBody(Cartesian pos, int time): assigns (pos.xx, pos.yy) to Cartesian position, executes position.concordanceDouble(), executes velocity.atOrigin(), executes Position(), sets bool \_at\_rest to TRUE, and assigns 0 to int \_n and time to int \_t.

InertBody(int a, int b, int vx, int vy): assigns (a, b) to Cartesian position, executes position.concordanceInt(), assigns (vx, vy) to Cartesian velocity, executes velocity.concordanceInt(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, executes Position(), and assigns 0 to int \_n and \_DEFAULT\_TIME to int \_t.

InertBody(double aa, double bb, double vxx, double vyy): assigns (aa, bb) to Cartesian position, executes position.concordanceDouble(), assigns (vxx, vyy) to Cartesian velocity, executes velocity.concordanceDouble(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, executes Position(), and assigns 0 to int \_n and \_DEFAULT\_TIME to int \_t.

InertBody(Cartesian pos, Cartesian vel): assigns (pos.xx, pos.yy) to Cartesian position, executes position.concordanceDouble(), assigns (vel.xx, vel.yy) to Cartesian velocity, executes velocity.concordanceDouble(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, executes Position(), and assigns 0 to int \_n and \_DEFAULT\_TIME to int \_t.

InertBody(int a, int b, int vx, int vy, int time): assigns (a, b) to Cartesian position, executes position.concordanceInt(), assigns (vx, vy) to Cartesian velocity, executes velocity.concordanceInt(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, executes Position(), and assigns 0 to int \_n and time to int \_t.

InertBody(double aa, double bb, double vxx, double vyy, int time): assigns (aa, bb) to Cartesian position, executes position.concordanceDouble(), assigns (vxx, vyy) to Cartesian velocity, executes velocity.concordanceDouble(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, executes Position(), and assigns 0 to int \_n and time to int \_t.

InertBody(Cartesian pos, Cartesian vel, int time): assigns (pos.xx, pos.yy) to Cartesian position, executes position.concordanceDouble(), assigns (vel.xx, vel.yy) to Cartesian velocity, executes velocity.concordanceDouble(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, executes Position(), and assigns 0 to int \_n and time to int \_t.

InertBody(int a, int b, int vx, int vy, int ix, int iy): assigns (a, b) to Cartesian position, executes position.concordanceInt(), assigns (vx, vy) to Cartesian velocity, executes velocity.concordanceInt(), assigns (ix, iy) to Cartesian initialPosition, executes initialPosition.concordanceInt(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, and assigns 0 to int \_n and \_DEFAULT\_TIME to int \_t.

InertBody(double aa, double bb, double vxx, double vyy, double ixx, double iyy): assigns (aa, bb) to Cartesian position, executes position.concordanceDouble(), assigns (vxx, vyy) to Cartesian velocity, executes velocity.concordanceDouble(), assigns (ixx, iyy) to Cartesian initialPosition, executes initialPosition.concordanceDouble(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, and assigns 0 to int \_n and \_DEFAULT\_TIME to int \_t.

InertBody(Cartesian pos, Cartesian vel, Cartesian ipos): assigns (pos.xx, pos.yy) to Cartesian position, executes position.concordanceDouble(), assigns (vel.xx, vel.yy) to Cartesian velocity, executes velocity.concordanceDouble(), assigns (ipos.xx, ipos.yy) to Cartesian initialPosition, executes initialPosition.concordanceDouble(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, and assigns 0 to int \_n and \_DEFAULT\_TIME to int \_t.

InertBody(int a, int b, int vx, int vy, int ix, int iy, int time): assigns (a, b) to Cartesian position, executes position.concordanceInt(), assigns (vx, vy) to Cartesian velocity, executes velocity.concordanceInt(), assigns (ix, iy) to Cartesian initialPosition, executes initialPosition.concordanceInt(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, and assigns 0 to int \_n and time to int \_t.

InertBody(double aa, double bb, double vxx, double vyy, double ixx, double iyy, int time): assigns (aa, bb) to Cartesian position, executes position.concordanceDouble(), assigns (vxx, vyy) to Cartesian velocity, executes velocity.concordanceDouble(), assigns (ixx, iyy) to Cartesian initialPosition, executes initialPosition.concordanceDouble(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, and assigns 0 to int \_n and time to int \_t.

InertBody(Cartesian pos, Cartesian vel, Cartesian ipos, int time): assigns (pos.xx, pos.yy) to Cartesian position, executes position.concordanceDouble(), assigns (vel.xx, vel.yy) to Cartesian velocity, executes velocity.concordanceDouble(), assigns (ipos.xx, ipos.yy) to Cartesian initialPosition, executes initialPosition.concordanceDouble(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, and assigns 0 to int \_n and time to int \_t.

InertBody(const InertBody &bodyCopy): (***copy constructor***) assigns (bodyCopy.position.xx, bodyCopy.position.yy) to Cartesian position of current InertBody object, executes position.concordanceDouble(), assigns (bodyCopy.velocity.xx, bodyCopy.velocity.yy) to velocity of current InertBody object, executes velocity.concordanceDouble(), assigns (bodyCopy.initialPosition.xx, bodyCopy.initialPosition.yy) to Cartesian initialPosition, executes initialPosition.concordanceDouble(), creates Cartesian Zero.atOrigin(), sets bool \_at\_rest to TRUE if velocity and Zero are equal, resets it to FALSE otherwise, and assigns bodyCopy.\_n to int \_n and bodyCopy.\_t to int \_t.

*Operators:* ***Assignment*** (InertBody operator=(const InertBody &bodyAssignment)): Same as ***copy constructor***, but checks for self-assignment, for which case nothing happens, and uses bodyAssignment to copy from instead of bodyCopy, and at the end, the current object (\*this) is returned.

***Equality***(bool operator==(const InertBody &bodyEqual1, const InertBody &bodyEqual2)): assigns false to a new bool equality, compares Cartesian bodyEqual1.position and Cartesian bodyEqual2.position to see if they are equal (using class Cartesian’s equality operator). If true, Cartesian bodyEqual1.velocity and Cartesian bodyEqual2.velocity are compared to see if they are equal. If true, Cartesian bodyEqual1.initialPosition and Cartesian bodyEqual2.initialPosition are compared to see if they are equal. If true, true is assigned to equality. Otherwise, if any of these tests fail, false remains assigned to equality. Finally, equality is returned.

***Inequality***(bool operator!=(const InertBody &bodyUnequal1, const InertBody &bodyUnequal2)): assigns false to a new bool inequality, and compares Cartesian bodyUnequal1.position and Cartesian bodyUnequal2.position to see if they are unequal. If so, true is assigned to inequality. Otherwise, Cartesian bodyUnequal1.velocity is compared to Cartesian bodyUnequal2.velocity to see if they are unequal. If so, true is assigned to inequality. Otherwise, Cartesian bodyUnequal1.initialPosition is compared to Cartesian bodyUnequal2.initialPosition to see if they are unequal. If so, true is assigned to inequality. Otherwise, if all these tests fail, false remains assigned to inequality. Finally, inequality is returned.

*Destructor:* ~InertBody(void): deletes Cartesian position, Cartesian velocity, and Cartesian initialPosition before rest of current object is destroyed.

*Functions that read the position of the current object:*

int X(void): returns int position.x.

int Y(void): returns int position.y.

double XX(void): returns double position.xx.

double YY(void): returns double position.yy.

Cartesian Position(void): returns Cartesian position.

*Functions that assign the position of the current object:*

void X(int a): assigns a to int position.x and executes position.concordanceInt().

void Y(int b): assigns b to int position.y and executes position.concordanceInt().

void X\_Y(int a, int b): assigns a to int position.x, assigns b to int position.y, and executes position.concordanceInt().

void XX(double aa): assigns aa to double position.xx and executes position.concordanceDouble().

void YY(double bb): assigns bb to double position.yy and executes position.concordanceDouble().

void XX\_YY(double aa, double bb): assigns aa to double position.xx, assigns bb to double position.yy, and executes position.concordanceDouble().

void Position(Cartesian pos): assigns double pos.xx to double position.xx, assigns double pos.yy to double position.yy, and executes position.concordanceDouble().

void Position(void): assigns values of Cartesian position object to Cartesian initialPosition object by assigning double position.xx to double initialPosition.xx, assigning double position.yy to double initialPosition.yy, and executing initialPosition.concordanceDouble().

*Functions that read the velocity of the current object:*

int VX(void): returns int velocity.x.

int VY(void): returns int velocity.y.

double VXX(void): returns double velocity.xx.

double VYY(void): returns double velocity.yy.

Cartesian Velocity(void): returns Cartesian velocity

bool AtRest(void): returns bool \_at\_rest

*Functions that assign the velocity of the current object:*

void VX(int vx): assigns vx to int velocity.x and executes velocity.concordanceInt().

void VY(int vy): assigns vy to int velocity.y and executes velocity.concordanceInt().

void VX\_VY(int vx, int vy): assigns vx to int velocity.x, assigns vy to int velocity.y, and executes velocity.concordanceInt().

void VXX(double vxx): assigns vxx to double velocity.xx and executes velocity.concordanceDouble().

void VYY(double vyy): assigns vyy to double position.yy and executes position.concordanceDouble().

void VXX\_VYY(double vxx, double vyy): assigns vxx to double velocity.xx, assigns bb to double velocity.yy, and executes velocity.concordanceDouble().

void Velocity(Cartesian vel): assigns double vel.xx to double velocity.xx, assigns double vel.yy to double velocity.yy, and executes velocity.concordanceDouble().

*Functions that read the initial position of the current object:*

int initX(void): returns int initialPosition.x.

int initY(void): returns int initialPosition.y.

double initXX(void): returns double initialPosition.xx.

double initYY(void): returns double initialPosition.yy.

Cartesian initPosition(void): returns Cartesian initialPosition.

*Functions that assign the initial position of the current object:*

void initX(int ix): assigns ix to int initialPosition.x and executes initialPosition.concordanceInt().

void initY(int iy): assigns iy to int initialPosition.y and executes initialPosition.concordanceInt().

void initX\_Y(int ix, int iy): assigns ix to int initialPosition.x, assigns iy to int initialPosition.y, and executes initialPosition.concordanceInt().

void initXX(double ixx): assigns ixx to double initialPosition.xx and executes initialPosition.concordanceDouble().

void initYY(double iyy): assigns iyy to double initialPosition.yy and executes initialPosition.concordanceDouble().

void initXX\_YY(double ixx, double iyy): assigns ixx to double initialPosition.xx, assigns iyy to double initialPosition.yy, and executes initialPosition.concordanceDouble().

void initPosition(Cartesian ipos): assigns double ipos.xx to double initialPosition.xx, assigns double ipos.yy to double initialPosition.yy, and executes initialPosition.concordanceDouble().

void initPosition(void): assigns values of Cartesian initialPosition object to Cartesian position object by assigning double initialPosition.xx to double position.xx, assigning double initialPosition.yy to double position.yy, and executing position.concordanceDouble().

void posCurInitJux(void): juxtaposes the values of Cartesian position and Cartesian initialPosition by creating new Cartesian jux and executing jux.atOrigin(), assigning double position.xx to double jux.xx and double position.yy to jux.yy, assigning double initialPosition.xx to position.xx and double iniialPosition.yy to position.yy, executing position.concordanceDouble(), assigning jux.xx to initialPosition.xx and jux.yy to initialPosition.yy, executing initialPosition.concodanceDouble(), and finally deleting jux.

*Functions that assign the next position in the path of the current object and then return it:*

int nextX(void): increments int \_n, if position.concordance() returns false, executes position.concordanceInt(), then if velocity.concordance() returns false, executes velocity.concordanceInt(), creates a new double tt and assigns (double) \_n / (double) int \_t to it, assigns double initialPosition.xx + (double velocity.xx \* tt) to double position.xx, executes position.concordanceDouble(), and returns int position.x.

int nextY(void): increments int \_n, if position.concordance() returns false, executes position.concordanceInt(), then if velocity.concordance() returns false, executes velocity.concordanceInt(), creates a new double tt and assigns (double) \_n / (double) int \_t to it, assigns double initialPosition.yy + (double velocity.yy \* tt) to double position.yy, executes position.concordanceDouble(), and returns int position.y.

double nextXX(void): increments int \_n, creates a new double tt and assigns (double) \_n / (double) int \_t to it, assigns double initialPosition.xx + (double velocity.xx \* tt) to double position.xx, executes position.concordanceDouble(), and returns position.xx.

double nextYY(void): increments int \_n, creates a new double tt and assigns (double) \_n / (double) int \_t to it, assigns double initialPosition.yy + (double velocity.yy \* tt) to double position.yy, executes position.concordanceDouble(), and returns position.yy.

Cartesian nextPosition(void): increments int \_n, creates a new double tt and assigns (double) \_n / (double) int \_t to it, assigns double initialPosition.xx + (double velocity.xx \* tt) to double position.xx, assigns double initialPosition.yy + (double velocity.yy \* tt) to double position.yy, executes position.concordanceDouble(), and returns Cartesian position.

*Functions that assign the previous position in the path of the current object and then return it:*

int prevX(void): decrements int \_n, if position.concordance() returns false, executes position.concordanceInt(), then if velocity.concordance() returns false, executes velocity.concordanceInt(), creates a new double tt and assigns (double) \_n / (double) int \_t to it, assigns double initialPosition.xx + (double velocity.xx \* tt) to double position.xx, executes position.concordanceDouble(), and returns int position.x.

int prevY(void): decrements int \_n, if position.concordance() returns false, executes position.concordanceInt(), then if velocity.concordance() returns false, executes velocity.concordanceInt(), creates a new double tt and assigns (double) \_n / (double) int \_t to it, assigns double initialPosition.yy + (double velocity.yy \* tt) to double position.yy, executes position.concordanceDouble(), and returns int position.y.

double prevXX(void): decrements int \_n, creates a new double tt and assigns (double) \_n / (double) int \_t to it, assigns double initialPosition.xx + (double velocity.xx \* tt) to double position.xx, executes position.concordanceDouble(), and returns position.xx.

double prevYY(void): decrements int \_n, creates a new double tt and assigns (double) \_n / (double) int \_t to it, assigns double initialPosition.yy + (double velocity.yy \* tt) to double position.yy, executes position.concordanceDouble(), and returns position.yy.

Cartesian prevPosition(void): decrements int \_n, creates a new double tt and assigns (double) \_n / (double) int \_t to it, assigns double initialPosition.xx + (double velocity.xx \* tt) to double position.xx, assigns double initialPosition.yy + (double velocity.yy \* tt) to double position.yy, executes position.concordanceDouble(), and returns Cartesian position.

*Functions that return and assign protected members of the current object:*

int TimeSlice(void): return int \_t.

int NumberOfScices(void): return int \_n.

void TimeSlice(int time): assign time to int \_t.

void NumberOfSlices(int num): assign num to int \_n.

class GravitationalBody

The class GravitationalBody is a derived class of class InertBody and has all the members from it (public: Cartesian position, Cartesian velocity, Cartesian initialPosition; and protected: bool \_at\_rest, int \_n, and int \_t) plus its own members (public: Cartesian initialVelocity; and protected: int \_ground, double \_gground, long int \_accgrav, double \_aaccgrav, int \_cor100, double \_ccor, bool \_groundOn, and bool \_gravOn). Whereas objects of class InertBody do not have any acceleration applied to them, objects of class GravitationalBody do. Although this acceleration can be any number directly encoded into long int \_accgrav or double \_aaccgrav, or by using the functions \_G(long int) or \_G(double), typically it would be one of the predefined constants below for terrestrial gravitational acceleration, or by using one of the functions that assigns them.

*Constants defined by precompiler:*

GSI is 9.80665 (Standard gravity on Earth in SI units = 9.80665 m/sec²)

GCGS is 980.665 (Standard gravity on Earth in CGS units = 980.665 cm/sec²)

GMS2 is 9 806 650 (Standard gravity on Earth in microns/sec² = 9,806,650 microns/sec²)

GUS is 32.1740 (Standard gravity on Earth in US units = 32.1740 ft/sec²)

GIS2 is 386.088 (Standard gravity on Earth in inches/sec² = 386.088 in/sec²)

G1000IS2 is 386 088 (Standard gravity on Earth in 1000ths of inches/sec² = 386,088 in/1000sec²)

*Variable Member:* Cartesian initialVelocity.

**Protected** *Variable Members:*

int \_ground, double \_gground (horizontal y/yy-level ground)

long int \_accgrav, double \_aaccgrav (acceleration of gravity)

int \_cor100 (percentage of coefficient of restitution—bounceback from ground)

double \_ccor (same as int \_cor100, but expressed as a pure decimal)

bool \_groundOn (switches horizontal ground on or off)

bool \_gravOn (switches the gravity on or off)

**Inherited** *Variable Members:* Cartesian objects: position, velocity, initialPosition (inherited from class InertBody).

**Inherited Protected** *Variable Members:* bool \_at\_rest, ints: \_n, \_t (inherited from class InertBody).

*Constructors:*

// constructors

GravitationalBody(void);

GravitationalBody(int a, int b);

GravitationalBody(double aa, double bb);

GravitationalBody(Cartesian pos);

GravitationalBody(int a, int b, int vx, int vy);

GravitationalBody(double aa, double bb, double vxx, double vyy);

GravitationalBody(Cartesian pos, Cartesian vel);

GravitationalBody(int a, int b, int vx, int vy, int ix, int iy);

GravitationalBody(double aa, double bb, double vxx, double vyy, double ixx, double iyy);

GravitationalBody(Cartesian pos, Cartesian vel, Cartesian ipos);

GravitationalBody(int a, int b, int vx, int vy, int ix, int iy, int ivx, int ivy);

GravitationalBody(double aa, double bb, double vxx, double vyy, double ixx, double iyy, double ivxx, double ivyy);

GravitationalBody(Cartesian pos, Cartesian vel, Cartesian ipos, Cartesian ivel);

GravitationalBody(int a, int b, int vx, int vy, int ix, int iy, int ivx, int ivy, int grnd);

GravitationalBody(double aa, double bb, double vxx, double vyy, double ixx, double iyy, double ivxx, double ivyy, double ggrnd);

GravitationalBody(Cartesian pos, Cartesian vel, Cartesian ipos, Cartesian ivel, double ggrnd);

GravitationalBody(int a, int b, int vx, int vy, int ix, int iy, int ivx, int ivy, int grnd, int agv);

GravitationalBody(double aa, double bb, double vxx, double vyy, double ixx, double iyy, double ivxx, double ivyy, double ggrnd, double aagv);

GravitationalBody(Cartesian pos, Cartesian vel, Cartesian ipos, Cartesian ivel, double ggrnd, double aagv);

GravitationalBody(int a, int b, int vx, int vy, int ix, int iy, int ivx, int ivy, int grnd, int agv, int epsln100);

GravitationalBody(double aa, double bb, double vxx, double vyy, double ixx, double iyy, double ivxx, double ivyy, double ggrnd, double aagv, double epsilon);

GravitationalBody(Cartesian pos, Cartesian vel, Cartesian ipos, Cartesian ivel, double ggrnd, double aagv, double epsilon);

GravitationalBody(const GravitationBody &gravbodCopy);

GravitationalBody operator=(const GravitationalBody &gravbodAssignment);

class TerrestrialGravitation

**GNU Free Documentation License, Version 1.3, 3 November 2008**

[This license applies to copying, releasing, and editing this document.]

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<https://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word

processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover

Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material.

If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single

copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by

adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number.

Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules

of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a

copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that

document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights

of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/licenses/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this

License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

**GNU GENERAL PUBLIC LICENSE, Version 3, 29 June 2007**

[This license applies to using, copying, releasing, and changing the software library, TerrestrialGravitation, that accompanies this document.]

Copyright © 2007 Free Software Foundation, Inc. <<https://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

**Preamble**

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

**TERMS AND CONDITIONS**

**0. Definitions.**

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

**1. Source Code.**

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

**2. Basic Permissions.**

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

**3. Protecting Users' Legal Rights From Anti-Circumvention Law.**

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

**4. Conveying Verbatim Copies.**

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

**5. Conveying Modified Source Versions.**

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

• a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

• b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.

• c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

• d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

**6. Conveying Non-Source Forms.**

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

• a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

• b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

• c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

• d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

• e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

**7. Additional Terms.**

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

• a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

• b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

• c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

• d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

• e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

• f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

**8. Termination.**

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

**9. Acceptance Not Required for Having Copies.**

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

**10. Automatic Licensing of Downstream Recipients.**

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

**11. Patents.**

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's “contributor version”.

A contributor's “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

**12. No Surrender of Others' Freedom.**

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

**13. Use with the GNU Affero General Public License.**

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

**14. Revised Versions of this License.**

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

**15. Disclaimer of Warranty.**

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

**16. Limitation of Liability.**

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

**17. Interpretation of Sections 15 and 16.**

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

**How to Apply These Terms to Your New Programs**

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>  
 Copyright (C) <year> <name of author>  
  
 This program is free software: you can redistribute it and/or modify  
 it under the terms of the GNU General Public License as published by  
 the Free Software Foundation, either version 3 of the License, or  
 (at your option) any later version.  
  
 This program is distributed in the hope that it will be useful,  
 but WITHOUT ANY WARRANTY; without even the implied warranty of  
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
 GNU General Public License for more details.  
  
 You should have received a copy of the GNU General Public License  
 along with this program. If not, see <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author>  
 This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.  
 This is free software, and you are welcome to redistribute it  
 under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <<https://www.gnu.org/licenses/>>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <<https://www.gnu.org/licenses/why-not-lgpl.html>>.

1. Although technically geometric vectors are movable and generally do not denote their locations, they nonetheless have two components and, if one assumes that they begin at the origin (0, 0), they then serve a very similar function to coordinates and complex numbers. [↑](#footnote-ref-1)