Naval Research Laboratory

Washington, DC 20375-5320

NRL/MR/5753--20-10,193

Algorithm Analyses for Electronic Warfare Simulation Over a Curved Earth

Donald E. Jarvis

Advanced Techniques Branch
Tactical Electronic Warfare Division

Justin N. Manning

Luke J. Prince

Applied Technology, Inc.
King George, VA

November 19, 2020

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
19-11-2020	NRL Memorandum Report	10/2018 – 10/2019

4. TITLE AND SUBTITLE

Algorithm Analyses for Electronic Warfare Simulation Over a Curved Earth

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Donald E. Jarvis, Justin N. Manning*, and Luke J. Prince*

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
6B46

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/5753--20-10,193

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375-5320

10. SPONSOR / MONITOR'S ACRONYM(S)
NRL

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

*Applied Technology, Inc., 5200 Potomac Dr., King George, VA, 22485

14. ABSTRACT

This report collects results of analysis carried out in the effort to bring an intrinsic curved earth model into the NextGen CRUISE_Missiles high fidelity modeling and simulation tool. These include a summary of software design considerations, the analytical results underlying the specular point algorithm, and an 'apples-to-apples' comparison of definitions of approximate equality that appear in different software libraries and are relevant to automated testing. These results document implemented algorithms and are expected to be relevant to other electronic warfare applications.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Donald E. Jarvis
Unclassified Unlimited	Unclassified Unlimited	Unclassified Unlimited	Unclassified Unlimited	16	**19b. TELEPHONE NUMBER** *(include area code)* (202) 404-7682

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

This page intentionally left blank.

Contents

This page intentionally left blank.

1 Introduction

This report collects results of analysis carried out in support of bringing an intrinsic curved earth model into the NextGen CRUISE_Missiles (NGCM) high fidelity modeling and simulation (M&S) tool. The results serve to document implemented algorithms and are expected to be relevant to other electronic warfare applications.

The key principle of our technical approach to introducing an intrinsic curved earth model was to 1) identify locations in the codebase with earth surface-related calculations, and 2) refactor the codebase to migrate these calculations to a new Earth Surface software object. Among other things, this involved the introduction of a key conceptual distinction: previously, the base coordinate frame and the earth surface were conflated (the earth surface and the *x*-*y* plane of the base coordinate system were one and the same); our changes required separating out the base coordinate frame and the earth surface as distinct roles.

Different Earth Surface object implementations can model different earth surface shapes. For development and testing, our plan was to roll these out according to the following strategy: first a planar earth, to preserve legacy behavior; then a spherical earth, the simplest curved surface, to support the exposure and elimination of implicit flat-earth assumptions throughout the codebase while benefiting from geometry algorithms that are as simple as possible; and finally an oblate spheroid, the class which includes WGS84 but many of whose algorithms are distinctly more complex than those of the sphere.

Manuscript approved November 16, 2020.

2 Specular Point Calculation

The calculations for high fidelity modeling and simulation of electronic warfare interactions over a curved earth will often include solving for the location of a specular point. This is the point where an optical ray reflects from the earth's surface on the way from a transmitter to a receiver.

Solving for a specular point over a flat surface is straightforward, but over a curved surface it is not. Even the problem of finding the specular point on a circle is more difficult than it appears: an exact solution involves solving a quartic equation [Miller1993]. An insightful geometrical interpretation of the quartic's four solutions is given in the appendix of [Kollas2017]. Much EW work in the past has relied on a cubic equation approximation due to Fishback ([Kerr], [Blake]), but this approximation has unknown error bounds. The problem of computing specular points on more general curved surfaces arises in radar cross section calculations [Gordon2010], [Gordon2014]. Specular points on an ellipsoid such as the WGS84 model of earth's surface arise in satellite remote sensing, and algorithms for computing them are of continuing interest [Southwell2018].

In this section we derive an algorithm that solves for the specular point on a curved surface. The algorithm essentially consists of a standard multivariate equation solver, and the equations come directly from geometrical optics. One advantage of the approach taken here is that it is generic, i.e., it is not hardcoded to a particular surface, thus supporting calculations on a variety of curved surface models. Among other things, from a software engineering viewpoint this makes the algorithm more testable.

2.1 Setting up the equations

The equations to be solved will come from geometrical optics but the choice is not unique. Some possibilities considered included:

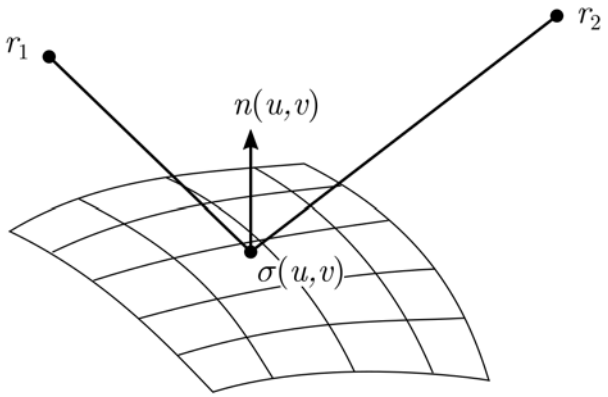
- Fermat's principle of least time. This approach leads to a minimization problem, which is a reasonable approach, but engineering judgment suggested that an equation solver would have better runtime performance than an optimizer.
- The reflection law in 3D: the incident unit vector, reflected through the tangent plane at the specular point, will equal the exiting unit vector from the specular point to the receiver. This approach is in a sense equivalent to what we ended up using (described next), but the basic equation solvers are not well suited to a problem set up as one equation (the scalar error between the reflected and exiting vectors) with two unknowns (the coordinate u, v).

Among the laws of reflection in geometrical optics we find the following ([Hecht] section 4.2.3) which we use in our algorithm:

- equal angles: the angle of incidence equals the angle of reflection

- in-plane: the directions of the incident ray, reflected ray, and surface normal are all coplanar

We express the conditions for a specular point on a curved surface in terms of these laws. Let the curved surface be defined by a function σ that takes a pair of scalar coordinates u, v (e.g. these may represent latitude and longitude) to a 3D Cartesian point, and let the function n return the unit normal vector at the same coordinates (to reduce notational clutter, we will freely leave the explicit dependence of σ and n on u, v unwritten). Let the fixed Cartesian points r_1, r_2 give the source and receiver locations.



Normalizations of $\sigma - r_1$ and $r_2 - \sigma$ represent the incident and exiting rays, respectively. For the equal angles law it is adequate to require that the incident and exiting angles have equal cosines, which (using the relationship between the cosine and the dot product) can be expressed as

$$\frac{\sigma - r_1}{\left\|\sigma - r_1\right\|} \cdot n = \frac{r_2 - \sigma}{\left\|r_2 - \sigma\right\|} \cdot n$$

To express the in-plane law we use the scalar triple product, which is a common way to express coplanarity.

$$\left(\frac{\sigma - r_1}{\left\|\sigma - r_1\right\|} \times \frac{r_2 - \sigma}{\left\|r_2 - \sigma\right\|} \right) \cdot n = 0$$

The resulting system of equations to be solved is

$$f\left(u,v\right) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} \left(\frac{\sigma - r_1}{\left\|\sigma - r_1\right\|} - \frac{r_2 - \sigma}{\left\|r_2 - \sigma\right\|} \right) \cdot n \\ \left(\frac{\sigma - r_1}{\left\|\sigma - r_1\right\|} \times \frac{r_2 - \sigma}{\left\|r_2 - \sigma\right\|} \right) \cdot n \end{bmatrix} = 0$$

where the equal angles law is captured by $f_1 = 0$ and the in-plane law by $f_2 = 0$. When parameters u, v are found that satisfy the above equation, then the specular point has parametric coordinates u, v and Cartesian coordinates $\sigma\left(u, v\right)$.

2.2 Solver

As with the physical laws, there are several choices for numerical equation solvers. One alternative that was considered but ultimately not used was the class of Broyden solvers [Sauer]. As multivariate 'secant' methods, the Broyden solvers have the attractive feature of being derivative-free, i.e., the derivatives (specifically, the Jacobian matrix of partial derivatives) do not have to be provided to the algorithm as a subroutine. Both handwritten (based on [Sauer]) and scientific library implementations (python scipy) of Broyden solvers were applied to our sample problems, and they performed poorly. This experience motivated the move to a multivariate Newton solver.

2.2.1 Initialization

The multivariate Newton's method requires an initial guess. Roughly speaking, this is done by invoking a flat earth approximation. If the source and destination points r_1, r_2 (vectors) are at heights h_1, h_2 above the surface (scalars), we compute the vector

$$\frac{h_2 r_1 + h_1 r_2}{h_1 + h_2}$$

The projection of this vector onto the reflecting surface is the initial guess for the specular point. (Note that if the reflecting surface is a plane, this is an exact solution.)

The initial guess above is based on the following argument. Assume the reflecting surface is the plane $z = 0$. It follows immediately from Fermat's principle that the specular point $\begin{bmatrix} x & y & 0 \end{bmatrix}^T$ for source and receiver locations $r_1 = \begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix}^T$ and $r_2 = \begin{bmatrix} x_2 & y_2 & z_2 \end{bmatrix}^T$ coincides with the surface intersection point between r_1 and the reflection $\begin{bmatrix} x_2 & y_2 & -z_2 \end{bmatrix}^T$ of r_2. We set up an equation to solve for this intermediate surface intersection point:

$$\alpha \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \left(1 - \alpha\right) \begin{bmatrix} x_2 \\ y_2 \\ -z_2 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

The constraint to be solved is the final 'row' of the equation above, namely,

$$\alpha z_1 - \left(1 - \alpha\right) z_2 = 0$$

from which we conclude $\alpha = \dfrac{z_2}{z_1 + z_2}$ and $1 - \alpha = \dfrac{z_1}{z_1 + z_2}$. Now, if we form

$$\alpha r_1 + \left(1 - \alpha\right) r_2 = \frac{z_2 r_1 + z_1 r_2}{z_1 + z_2}$$

and project the result to the surface (i.e., set its z component to zero), we recover the desired point $\begin{bmatrix} x & y & 0 \end{bmatrix}^T$. So to form the initial guess above, we use this formula, except with height above the surface taking the place of *z* coordinates (the *z* coordinate indicates height in the case of the the *x*-*y* plane, but not in general). Thus we arrive at an initial guess that is defined for both flat and curved surfaces, and reduces to the exact solution in the special case of a flat surface.

2.2.2 Algorithm

Once the derivative J is available (its calculation is detailed in the following section), the system of equations is solved using a straightforward implementation (following [Sauer]) of the multivariate Newton's method (since the Jacobian is only 2 by 2, the Gaussian elimination refinement was not used). The following update rule is iterated until the error is acceptably small:

$$\begin{bmatrix} u_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} u_k \\ v_k \end{bmatrix} - J^{-1} f\left(u_k, v_k\right)$$

2.3 The Derivative

The calculation of the Jacobian matrix needed by the multivariate Newton's method is given in this section. It is convenient to decompose the problem into computing the rows of the matrix, each of which is a (transposed) gradient:

$$J = \begin{bmatrix} \partial f_1 / \partial u & \partial f_1 / \partial v \\ \partial f_2 / \partial u & \partial f_2 / \partial v \end{bmatrix} = \begin{bmatrix} \left(\nabla f_1\right)^T \\ \left(\nabla f_2\right)^T \end{bmatrix}$$

The calculations that follow make frequent reference to derivative rules that are summarized in a later section.

2.3.1 Derivative of equal-angles law

Applying the dot product rule, the derivative ∇f_1 of the equal-angles criterion f_1 is:

$$\begin{aligned} \nabla f_1 &= \nabla \left(\frac{\sigma - r_1}{\left\| \sigma - r_1 \right\|} + \frac{\sigma - r_2}{\left\| \sigma - r_2 \right\|} \right)^T n \\ &= \left(\nabla \left(\frac{\left(\sigma - r_1\right)^T}{\left\| \sigma - r_1 \right\|} + \frac{\left(\sigma - r_2\right)^T}{\left\| \sigma - r_2 \right\|} \right) \right) n + \left(\nabla n^T \right) \left(\frac{\sigma - r_1}{\left\| \sigma - r_1 \right\|} + \frac{\sigma - r_2}{\left\| \sigma - r_2 \right\|} \right) \end{aligned}$$

The derivative in the first term of ∇f_1 has two terms of the same form. To a representative term we apply the normalized vector rule, and the fact that r is constant so $\nabla r = 0$:

$$\nabla \frac{\left(\sigma - r\right)^T}{\left\|\sigma - r\right\|} = \nabla\left(\sigma - r\right)^T \left(\frac{I}{\left\|\sigma - r\right\|} - \frac{\left(\sigma - r\right)\left(\sigma - r\right)^T}{\left\|\sigma - r\right\|^3} \right)$$
$$= \nabla\sigma^T \left(\frac{I}{\left\|\sigma - r\right\|} - \frac{\left(\sigma - r\right)\left(\sigma - r\right)^T}{\left\|\sigma - r\right\|^3} \right)$$

Therefore the derivative in the first term of ∇f_1 is

$$\nabla\sigma^T \left(\frac{I}{\left\|\sigma - r_1\right\|} + \frac{I}{\left\|\sigma - r_2\right\|} - \frac{\left(\sigma - r_1\right)\left(\sigma - r_1\right)^T}{\left\|\sigma - r_1\right\|^3} - \frac{\left(\sigma - r_2\right)\left(\sigma - r_2\right)^T}{\left\|\sigma - r_2\right\|^3} \right)$$

and so the first term of ∇f_1 is

$$\left(\frac{1}{\left\|\sigma - r_1\right\|} + \frac{1}{\left\|\sigma - r_2\right\|} \right)\left(\nabla\sigma^T\right)n - \frac{\left(\sigma - r_1\right)^T n}{\left\|\sigma - r_1\right\|^3}\left(\nabla\sigma^T\right)\left(\sigma - r_1\right) - \frac{\left(\sigma - r_2\right)^T n}{\left\|\sigma - r_2\right\|^3}\left(\nabla\sigma^T\right)\left(\sigma - r_2\right)$$

Note that $\nabla\sigma^T$ is (the transpose of) the Jacobian of the curved surface parameterization – not to be confused with the Jacobian of the optics laws over this curved surface, which we are currently deriving.

The only derivative to work out in the second term of ∇f_1 is ∇n^T. Note that the standard unit normal ([Pressley] pg. 127) is $n = \frac{\sigma_u \times \sigma_v}{\left\|\sigma_u \times \sigma_v\right\|}$. Then, again using the normalized vector rule,

$$\nabla n^T = \nabla \frac{\left(\sigma_u \times \sigma_v\right)^T}{\left\|\sigma_u \times \sigma_v\right\|}$$
$$= \nabla\left(\sigma_u \times \sigma_v\right)^T \left(\frac{1}{\left\|\sigma_u \times \sigma_v\right\|} I - \frac{\left(\sigma_u \times \sigma_v\right)\left(\sigma_u \times \sigma_v\right)^T}{\left\|\sigma_u \times \sigma_v\right\|^3} \right)$$

in which, by the cross product rule,

$$\nabla\left(\sigma_u \times \sigma_v\right)^T = \begin{bmatrix} \sigma_{uu} \times \sigma_v + \sigma_u \times \sigma_{uv} & \sigma_{uv} \times \sigma_v + \sigma_u \times \sigma_{vv} \end{bmatrix}^T$$

With this, we have the information needed to assemble an implementable formula for ∇f_1.

2.3.2 Derivative of in-plane law

Applying the dot product rule, the derivative ∇f_2 of the equal-angles criterion f_2 is:

$$\begin{aligned}\nabla f_2 &= \nabla\left(\frac{\sigma - r_1}{\|\sigma - r_1\|} \times \frac{\sigma - r_2}{\|\sigma - r_2\|}\right)^T n \\ &\triangleq \nabla {v_1}^T n \\ &= \left(\nabla n^T\right) v_1 + \left(\nabla {v_1}^T\right) n\end{aligned}$$

An expression for ∇n^T was computed above. It remains to evaluate $\nabla {v_1}^T$. By the cross product rule,

$$\begin{aligned}\nabla {v_1}^T &= \nabla\left(\frac{\sigma - r_1}{\|\sigma - r_1\|} \times \frac{\sigma - r_2}{\|\sigma - r_2\|}\right)^T \\ &= \begin{bmatrix} \left(\frac{\partial}{\partial u}\frac{\sigma - r_1}{\|\sigma - r_1\|} \times \frac{\sigma - r_2}{\|\sigma - r_2\|} + \frac{\sigma - r_1}{\|\sigma - r_1\|} \times \frac{\partial}{\partial u}\frac{\sigma - r_2}{\|\sigma - r_2\|}\right)^T \\ \left(\frac{\partial}{\partial v}\frac{\sigma - r_1}{\|\sigma - r_1\|} \times \frac{\sigma - r_2}{\|\sigma - r_2\|} + \frac{\sigma - r_1}{\|\sigma - r_1\|} \times \frac{\partial}{\partial v}\frac{\sigma - r_2}{\|\sigma - r_2\|}\right)^T \end{bmatrix}\end{aligned}$$

The partial derivatives in the expression above can be extracted from the gradients computed in the previous section.

$$\nabla \frac{\left(\sigma - r\right)^T}{\|\sigma - r\|} = \begin{bmatrix} \frac{\partial}{\partial u}\frac{\left(\sigma - r\right)^T}{\|\sigma - r\|} \\ \frac{\partial}{\partial v}\frac{\left(\sigma - r\right)^T}{\|\sigma - r\|} \end{bmatrix}$$

The above results can be combined to constitute an implementable formula for ∇f_2.

2.3.3 Note on the derivation

A brief attempt was made to compute this derivative with a symbolic calculator, but the prospects for expressing the result economically were not promising. The implementation of the above equations in working code was complemented with a unit test suite that checked the results of the formulas against numerical derivatives and canonical cases, to provide confidence in their correctness.

2.4 Matrix Calculus

Here we spell out some nuances of the semantics of the del operator "∇" as used in this report, and collect rules of matrix calculus that are referred to in taking the derivatives. Throughout this report we freely make use of subscript notation for partial derivatives to reduce notational clutter; for example, σ_u has the same meaning as $\partial\sigma/\partial u$.

The del operator takes the gradient with respect to u and v.

$$\nabla \triangleq \begin{bmatrix} \partial/\partial u \\ \partial/\partial v \end{bmatrix}$$

It can legally operate on a one-row vector; this includes a scalar as a special case.

$$\nabla a^T = \begin{bmatrix} a_u^T \\ a_v^T \end{bmatrix}$$

Del operates on the term to its right. This usage helps avoid excessive parentheses. E.g.,

$$\nabla ab + c = \left(\nabla ab\right) + c$$
$$\nabla\left(ab + c\right) = \left(\nabla ab\right) + \left(\nabla c\right) = \nabla ab + \nabla c$$

Derivative of a dot product:

$$\nabla a^T b = \left(\nabla b^T\right) a + \left(\nabla a^T\right) b$$

Derivative of a cross product:

$$\frac{\partial}{\partial u}\left(a \times b\right) = a_u \times b + a \times b_u$$
$$\nabla\left(a \times b\right)^T = \begin{bmatrix} a_u \times b + a \times b_u & a_v \times b + a \times b_v \end{bmatrix}^T$$

Derivative of magnitude:

$$\nabla \left\|a\right\| = \frac{\left(\nabla a^T\right) a}{\left\|a\right\|}$$

Derivative of a normalized vector:

$$\begin{aligned} \nabla \frac{a^T}{\left\|a\right\|} &= \frac{1}{\left\|a\right\|^2}\left(\left\|a\right\| \nabla a^T - \left(\nabla \left\|a\right\|\right) a^T\right) \\ &= \frac{\nabla a^T}{\left\|a\right\|} - \frac{\left(\nabla \left\|a\right\|\right) a^T}{\left\|a\right\|^2} \\ &= \frac{\nabla a^T}{\left\|a\right\|} - \frac{\left(\nabla a^T\right) a a^T}{\left\|a\right\|^3} \\ &= \nabla a^T \left(\frac{1}{\left\|a\right\|} I - \frac{a a^T}{\left\|a\right\|^3}\right) \end{aligned}$$

2.5 References

[Miller1993] A.R. Miller and E. Vegh, "Exact result for the grazing angle of specular reflection from a sphere," SIAM Review 35(3), Sept. 1993

[Kollas2017] N.K. Kollas, "Specular reflection on the surface of a sphere: compass and ruler constructions," https://arxiv.org/pdf/1703.06768.pdf

[Kerr] D.E. Kerr, ed., *Propagation of Short Radio Waves,* Boston Technical Publishers, 1964

[Blake] L.V. Blake, *Radar Range-Performance Analysis*, Artech House, 1986

[Gordon2010] W.B. Gordon, "Reflections From Multiple Surfaces Without Edges," IEEE Transactions on Antennas and Propagation 58(10), Oct. 2010, doi: 10.1109/TAP.2010.2055785

[Gordon2014] W. B. Gordon, "A Method for Locating Specular Points," IEEE Transactions on Antennas and Propagation 62(4), April 2014, doi: 10.1109/TAP.2014.2299271

[Southwell2018] B.J. Southwell and A.G. Dempster, "A New Approach to Determine the Specular Point of Forward Reflected GNSS Signals," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 11(2), Feb. 2018, doi: 10.1109/JSTARS.2017.2775647

[Hecht] E. Hecht and A. Zajac, *Optics*, Addison-Wesley, 1974

[Sauer] T. Sauer, *Numerical Analysis*, 2nd ed. Pearson, 2012

[Pressley] A. Pressley, *Elementary Differential Geometry*, Springer, 2001

3 Approximate Equality

In this section we analyze simple examples to identify the features of a useful definition of approximate equality, and then apply this understanding to definitions appearing in software libraries.

Numerous applications involving scientific or numerical codes involve checks on floating point quantities. In these checks we are almost never interested in exact equality. A concept of approximate equality is much more useful. Such a concept also arises in the determination of an acceptable error bound in various numerical solvers. However, approximate equality has to be defined. Two reasonable approaches to approximate equality are apparent:

- The absolute difference between the quantities has to be less than some threshold
- The quantities should be within some percent (or, equivalently, some dB) of each other

To discern which rule should be used, we consider some use cases. For example, we can envision an application where the numbers 10 and 0.01 should not be considered to be approximately equal to each other, but the whoppers 7,005,010.0 and 7,005,000.01 should be considered approximately equal to each other, and the iotas 1e-9 and 1e-12 should also be considered approximately equal to each other.

The absolute difference between the whoppers is equal to the absolute difference between 10 and 0.01. However, the whoppers are within a fraction of a percent of each other, whereas 10 and 0.01 are off by orders of magnitude. In this example a relative percentage-based rule encodes the desired notion of approximate equality.

However, when comparing the iotas, the percent error is irrelevant. Even though they are off by three orders of magnitude – just as 10 and 0.01 are – we want them to be considered approximately equal in this application because in an absolute sense they are both so small.

We conclude that a definition of approximate equality appropriate for scientific and numerical work will feature a relative tolerance rule for large quantities, an absolute tolerance rule for small quantities, and some means of determining which rule should apply.

3.1 Constructing a comparison expression

Generically, for two numbers a, b, the absolute error between them is expressed as $\left|a - b\right|$, and the relative error is

$$\frac{\left|a - b\right|}{\text{size}}$$

where 'size' is some measure of how big the numbers are. Different environments make different definitions of size.

Here we analyze several definitions of approximate equality used in current software libraries and compare them in an apples-to-apples way. This is intended to support applications such as the porting of unit tests between languages and environments.

3.2 Python math.isclose

The `math.isclose` expression (introduced in Python 3.5), defined in terms of a relative tolerance `rel_tol` and absolute tolerance `abs_tol`, is effectively:

```
abs(a-b) <= max( rel_tol*max(abs(a),abs(b)), abs_tol )
```

Here the size is defined as `max(abs(a),abs(b))`. Clearly, when the size (and/or the `rel_tol`) is small enough, the expression simplifies to

```
abs(a-b) <= abs_tol
```

which is an appropriate rule for comparing small quantities (such as the iotas above). Otherwise, the expression simplifies to

```
abs(a-b) <= rel_tol*max(abs(a),abs(b))
```

which is mathematically equivalent to

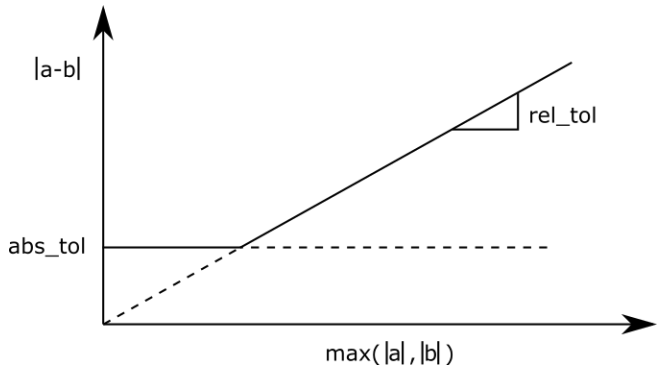
```
abs(a-b) / max(abs(a),abs(b)) <= rel_tol
```

We recognize the left hand side as a relative error ratio. This percentage-based rule is appropriate for comparing normal to large-sized quantities.

The `math.isclose` expression can be seen as checking whether the absolute difference is less than the greater, or more lenient, of the two thresholds. Equivalently, the expression switches between the absolute and relative rules at:

```
max(abs(a),abs(b)) == abs_tol / rel_tol
```

Significantly, both `abs_tol` and `rel_tol` can be specified by the user.



3.3 Python numpy.isclose

(Note that, while both are available in Python, the `numpy.isclose` function is not the same thing as the `math.isclose` function discussed above.)

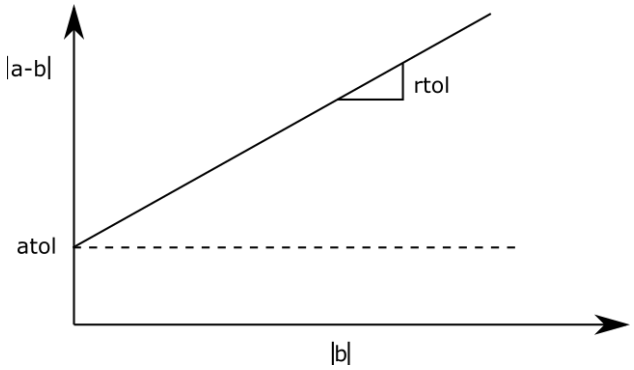
The `numpy.isclose` expression is:

```
absolute(a-b) <= (rtol*absolute(b) + atol)
```

Here the size is defined as `absolute(b)`. This effectively makes the second argument `b` the reference value. Unlike `math.isclose`, this expression is not symmetric in `a` and `b`.

There are still roles for absolute and relative thresholds, but (since the outer max of `math.isclose` is replaced with an addition) there is no switching point from one rule to the other. Presumably, this implementation trades away symmetry and rule-switching for faster execution.

Both `atol` and `rtol` can be specified by the user.



3.4 Google test

These tests are very stringent, and will often not be appropriate for applications such as checking for an adequate approximate match between different models.

- The `FLOAT_EQ`, `DOUBLE_EQ` macros evaluate to true only for values that are within a few ULPs (units in the last place) of each other. There are no options to pass in user-defined error thresholds.
- The `NEAR` macros take a user-specified absolute value error, but not a relative/percentage-based error.

Reference:
`https://github.com/abseil/googletest/blob/master/googletest/docs/advanced.md#floating-point-comparison` (visited September 30, 2019, commit `dc1ca9a`)

3.5 NGCM_Math::almostEquals

Reading the code, we see that the substance of this rule is:

```
abs(a-b) <= maxDiff OR abs(a-b) <= kMaxRelDiff*max(abs(a),abs(b))
```

which is mathematically equivalent to

```
abs(a-b) <= max( kMaxRelDiff*max(abs(a),abs(b)), maxDiff )
```

This expression is substantially the same as the python `math.isclose` rule above. However (in the current implementation):

- `maxDiff` is an optional parameter, but cannot be greater than the `kMaxDiff` value of 1e-15
- `kMaxRelDiff` is hardcoded to machine epsilon (there is no provision for a user-supplied parameter)

3.6 Rule of thumb for relative tolerance

A relative tolerance value of `5e-`*n* roughly corresponds to *n* digits of agreement (after rounding).

3.7 Conclusion

Some M&S applications, such as unit tests, or judging the agreement between two models of different levels of fidelity, require the ability for the analyst to specify a flexible definition of approximate equality. For such applications, an equality test with user-specified absolute and relative tolerances is preferred.