HASKELL SPACEFLIGHT WORKSHOP

Jonathan Merritt, Luke Clifton

YOW! LambdaJam 2019, May 13–15

Version timestamp: 2019-04-28T10:13:24+0000

Table of Contents

1 Introduction

In this workshop, we take an enthusiastic numerical approach to simulating spacecraft maneuvers. Our workshop examples appear in the published literature, yet we must begin by stating that there are often more elegant solutions to the problems we describe, allowing results to be found more economically and precisely. For these solutions, we must refer readers to a comprehensive textbook on astrodynamics (eg. [1]). We have chosen to take our somewhat simplified numerical approach for the following reasons:

1. It makes a 90 minute workshop possible, where an exploration of tailored methods would require weeks of work,
2. It allows us to frame several problems as initial value problems of ODE integration,
3. It introduces techniques that are widely applicable to dynamical systems outside of spaceflight, and
4. We believe it provides a good introduction, motivating more sophisticated approaches quite naturally as individuals explore further.

2
ODE Integration and Initial Value Problems

The models we use for a spacecraft depend upon a set of variables that represent its state at an instant in time. These state variables typically include:

- Position
- Velocity
- Mass

They may be scalar quantities or vectors, as appropriate to the problem.

Our simulations are all examples of "Initial Value Problems". In an initial value problem, we know the starting state of the spacecraft, and we have a set of first-order ordinary differential equations (ODEs), which describe how its state evolves with time. We will integrate these ODEs to predict the state at future times. Using this approach, we can compute the time history of state variables that are critial to mission or maneuver planning. For example, we might find the trajectory of a spacecraft (its position as a function of time), and check whether it places the spacecraft in a desired orbit.

The motion of a spacecraft depends on multiple forces that might be acting on it. For example:

- Gravity
- Atmospheric drag
- Rocket thrust

Thrust from a rocket engine may be controlled, and control inputs can be modeled easily in our system. Testing the behavior of a control system, particularly under conditions of real-world variations, is a modern practical use of the methods we cover (eg. [2, 3]).

2.1 Euler's Method

We can write a set of coupled, first-order ODEs as:

$$\frac{d\mathbf{x}}{dt} = \dot{\mathbf{x}} = f(t, \mathbf{x}) \tag{2.1}$$

Here, $\mathbf{x}$ is the state vector, t is time, and f is some function. In Euler's method, we approximate a step forward in time by multiplying the gradient by the size of the time step, h:

$$\begin{aligned} \mathbf{x}(t+h) &\approx \mathbf{x}(t) + \dot{\mathbf{x}}\, h && (2.2) \\ &\approx \mathbf{x}(t) + f\left(t, \mathbf{x}(t)\right)\, h && (2.3) \end{aligned}$$

2.1.1 Radioactive Decay

We will begin implementing Euler's method, specialized to `Double`, using the process of radioactive decay as an example. Radioactive decay is simple enough to solve analytically, thus providing a comparison with the numerical result, and it only involves a single state variable (N), which can be represented as a `Double`. Specializing to `Double` also gives us a simple starting point that is representative of the approach used in many other programming languages.

In radioactive decay, the rate of decay, $\dot{N}$, is proportional to the number of moles of radioactive particles that remain at any instant in time, N:

$$\dot{N} = -\lambda N \tag{2.4}$$

where λ is called the decay constant. This equation can be solved by knowing in advance that an exponential function happens to fit exactly the expected equation:

$$N = N_0 \exp\left(-\lambda t\right) \tag{2.5}$$

So that:

$$\begin{aligned} \dot{N} &= -\lambda\left(N_0 \exp\left(-\lambda t\right)\right) && (2.6) \\ &= -\lambda N && (2.7) \end{aligned}$$

as required. Conventionally, λ is specified in terms of the half-life of an isotope, $t_{\frac{1}{2}}$:

$$\text{at } t = 0,\ N = N_0 \tag{2.8}$$

$$\text{at } t = t_{\frac{1}{2}},\ N = \frac{N_0}{2} \tag{2.9}$$

thus:

$$\frac{N_0}{2} = N_0 \exp\left(-\lambda t_{\frac{1}{2}}\right) \tag{2.10}$$

$$\ln\left(\frac{1}{2}\right) = -\lambda t_{\frac{1}{2}} \tag{2.11}$$

$$\lambda = \frac{\ln 2}{t_{\frac{1}{2}}} \tag{2.12}$$

As an example, consider the isotope Plutonium-238 (^{238}Pu), which has been used in radioisotope thermoelectric generators (RTGs) for spacecraft such as the Voyager 1 and 2 probes. This isotope has a half-life of approximately 87.7 years.

Problem 1: Euler integration specialized to `Double`.

In the file `ODE.hs`,

- implement `eulerStepDouble`, which takes a single step of Euler integration
- implement `integrateEulerDouble`, which takes multiple steps

In `ODEExamples.hs`,

- run `plotEulerDoubleExpDecay Screen`, to view a plot of Euler integration applied to the radioactive decay example

Figure 2.1 shows the result of applying Euler integration to the radioactive decay example. In this figure, it is evident that when smaller time-steps are taken, the Euler method more closely approximates the analytical solution. This is usually the case practically with numerical integration, although there is a limit beyond which smaller time steps will begin to diverge from the correct solution due to accruing floating-point errors. We will see later that raising the polynomial order of the integration approximation can improve accuracy with greater computational efficiency than taking smaller time steps.

2.2 Generalizing Euler's Method

We will now generalize Euler's method using the abstractions available in the `vector-space` package. The necessary constraints are captured in Listing 1. Don't panic if this seems a lot to take in, since we'll see a few concrete examples.

The first concept we introduce is the difference between an `AffineSpace` and its associated `VectorSpace`. In the present context, points in the `AffineSpace` are points belonging to the state space of the problem, of type `state`. Vectors of the associated `VectorSpace`, of type `diff`, represent deltas or differences between the points. We can add a vector to a point to obtain a new point, but we don't sum points directly. Most widely-used frameworks for numerical integration do not make this distinction.

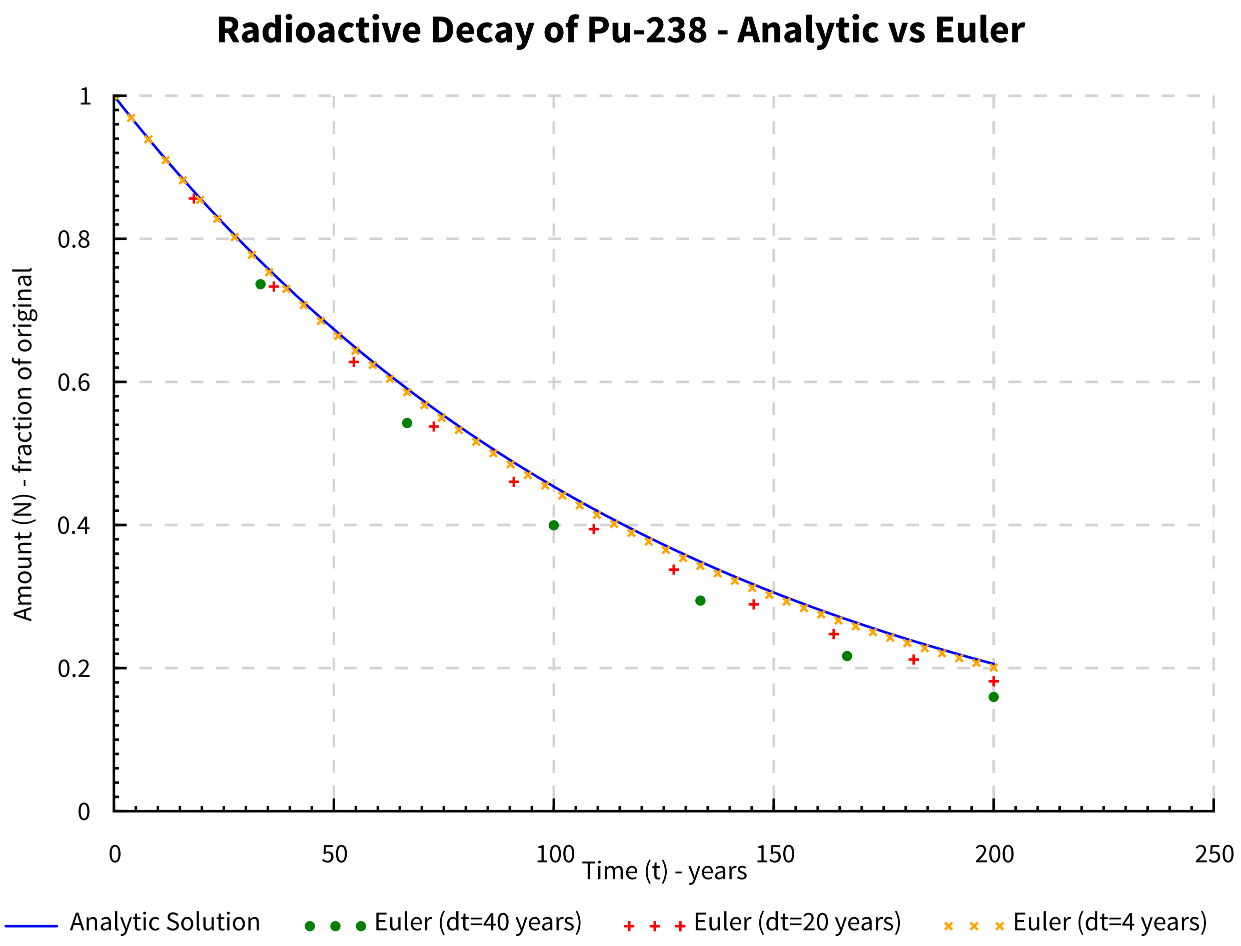

Figure 2.1: Comparison of Euler integration with the analytical result for radioactive decay of the isotope ^{238}Pu. The solid line shows the analytic solution while the points demonstrate the Euler approximation for different time steps.

```haskell
eulerStep
  :: ( AffineSpace state
     , diff ~ Diff state, VectorSpace diff
     , HasBasis time, HasTrie (Basis time)
     , s ~ Scalar diff, s ~ Scalar time )
  => time                                   -- ^ Step size @dt@
  -> ((time, state) -> time :-* diff)       -- ^ Gradient function @f (x, t)@
  -> (time, state)                          -- ^ Before the step @(t, x)@
  -> (time, state)                          -- ^ After the step @(t, x)@
```

Listing 1: Constraints for Euler's method generalized by `vector-space`.

Next is the concept of a linear map representing the co-domain of the gradient function: `time :-* diff`. In our case, where we deal with finite differences, this constructor can be considered analogous to a function of type `time -> diff`, but constrained to be linear. A very simple illustration of the linear map is shown in Listing 2. The `linear` function assumes that the function it has been provided is linear, and it memoizes the values of that function along each basis vector of the vector space.

```
> :set -XFlexibleContexts
> import Data.LinearMap ((:-*), linear, lapply)
> :{
| f :: Double -> Double
| f x = 5 * x
| :}
> linearMap = linear f
> :t linearMap
linearMap :: Double :-* Double
> lapply linearMap 1
5.0
> lapply linearMap 2
10.0
```

Listing 2: Illustration of a simple, scalar linear map.

Finally, we need to describe the operations that can be used on instances of `AffineSpace` and its associated `VectorSpace`:

`lapply m h`	applies linear map `m` to vector `h`
`p .+^ v`	adds vector `v` to point `p`
`a ^+^ b`	adds vector `a` to vector `b`

These operations are sufficient to implement the generalized form of Euler's method.

2.2.1 Simple Harmonic Motion

To motivate the generalized form of Euler's method, let's consider simple harmonic motion, with two scalar state variables:

- Position, r
- Velocity, v

The data type `State` in `ODEExamples.hs` describes this state, which is the `AffineSpace` of the problem. It introduces statically type-checked units from the `units` package for length and velocity. The data type `DState` is the correspondng `VectorSpace` of the problem, representing deltas in the state.

In simple harmonic motion, a linear spring force, F, is proportional to the position:

$$F = -kr \tag{2.13}$$

This is combined with the equations of motion of a point mass to produce the following governing ODE:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{r} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \\ -kr/m \end{bmatrix} \tag{2.14}$$

If the initial conditions of the problem at $t = 0$ are $r = r_0$ and $v = 0$ then the analytic solution is:

$$\mathbf{x}(t) = \begin{bmatrix} r_0 \cos(\omega t) \\ -\omega r_0 \sin(\omega t) \end{bmatrix} \tag{2.15}$$

where the angular velocity, ω, is:

$$\omega = \sqrt{\frac{k}{m}} \tag{2.16}$$

Problem 2: Generalized Euler integration.

In the file `ODE.hs`,

- implement `eulerStep`
- implement `integrate` and `integrateWithDiff`

Symbols

f	A function.
F	Total force, scalar (N).
h	Time step (s).
k	Spring constant (N/m).
m	Mass (kg).
N	Number of moles of a substance.
N_0	Number of moles of a substance at $t = 0$.
r	Position, scalar (m).
r_0	Position, scalar, at $t = 0$ (m).
t	Time (s).
$t_{\frac{1}{2}}$	Radioactive half life (s).
v	Velocity, scalar (m/s).
$\mathbf{x}$	System state vector.
λ	Radioactive decay constant (1/s).
ω	Angular frequency (1/s).

References

[1] R. H. Battin, *An introduction to the Mathematics and Methods of Astrodynamics, Revised Edition*. American Institute of Aeronautics and Astronautics, 1999.

[2] J. L. Prince, P. N. Desai, E. M. Queen, and M. R. Grover, "Mars phoenix entry, descent and landing simulation design and modelling analysis," *Journal of Spacecraft and Rockets*, vol. 48, no. 5, pp. 754–764, 2011. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080033126.pdf

[3] G. L. Brauer, D. E. Cornick, and R. Stevenson, *Capabilities and Applications of the Program to Optimize Simulated Trajectories (POST). Program Summary Document. NASA CR-2770.* NASA, 1977. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19770012832.pdf