Numerical Integration of Ordinary D.E.'s

I'll continue to use $x$ as dependent variable and $t$ as independent variable.

**State Space Form:**

$$\dot{x} = f(x, t) \quad x \text{ n-vector, func. of } t$$

Or write

$$\begin{cases} 
\frac{dx_1}{dt} = f_1(x_1, \ldots, x_n, t) \\
\frac{dx_2}{dt} = f_2(x_1, \ldots, x_n, t) \\
\vdots \\
\frac{dx_n}{dt} = f_n(x_1, \ldots, x_n, t)
\end{cases}$$

**Boundary Conditions** can be very complicated. Two simple cases cover most practical situations.

*Initial values given* $x_1(0), x_2(0), \ldots, x_n(0)$

$$\begin{array}{c}
\begin{array}{c}
x(0) \\
\downarrow \\
x(0)
\end{array}
\end{array}$$

*Two-point values given* usually some at $t=0$, some at $t=T_{\text{final}}$.

*We'll stick to initial-value problems.*

Often these arise from situations like

$$\frac{d^2y}{dt^2} + u(t) \frac{dy}{dt} + \beta(t) y = 0$$

& initial conditions $y(0)$ and $y'(0)$

(equivalent to $x_1(0)$ and $x_2(0)$ when converted to state-space)
We'll look at following general methods

1) Euler's Method (simple-minded, basis of many other methods, not good by itself)

2) predictor-corrector methods (can be excellent for reasonable problems)

3) Runge-Kutta (good workplace—will be good enough for usually implies 4th order) assignment 3, but not state-of-the-art)

See Nom. Recipes & refs therein for more sophisticated methods, like Richardson extrapolation. Integrating ODE's numerically is a grown-up problem!

Three criteria enter into evaluating a numerical integration method: (besides complexity of coding, of course)

1) accuracy
   - estimated using standard Taylor series arguments, \( O(h^2) \), \( O(h^3) \) ... etc.
   - plus empirical evidence

2) efficiency
   - running time can be difficult to predict, since we usually don't know \( h = \Delta t \) in advance, and it may be chosen adaptively in practice during run-time

3) stability
   - some methods diverge on some problems
Euler's Method ("point-slope")

Simplest, basis of others, but just a start
[Euler, *Institutiones Calculi Integralis*, 1768]

Starting point: \[ \dot{x} = f(x) \]

Remember that \( x \) & \( f \) are \( n \)-vectors, but that doesn't affect our results.

\[
x_0 = f(x_0) \approx \frac{x_1 - x_0}{h} \quad \Rightarrow \quad x_1 \approx x_0 + hf(x_0)
\]

This is the iteration: \[ x_1 := x_0 + hf(x_0) \]

---

Local (Single-Step) Error

Taylor's series again,

\[
x_1 = x_0 + hf(x_0) + \frac{h^2}{2} f'(\xi)
\]

Some \( \xi \in [t_0, t_1] \)

\[ E_{\text{single-step}} = x_1 - [x_0 + hf(x_0)] = O(h^2) \quad \text{as } h \to 0 \]

---

Global (Accumulated) Error

Integrating from \( t = 0 \) to \( T \), we have \( T/k \) steps.

Heuristically, \[ E_{\text{global}} = O(h) \] (can be made rigorous, see J.R. Rice, *Numerical Methods, Software, and Analysis*, 1966, McGraw-Hill, 1983)
Outline of Predictor-Corrector Idea

See [Acton70] for details

1) Use past values to predict \( \dot{x}_1 \) from \( \dot{x}_{-3}, \dot{x}_{-2}, \dot{x}_{-1}, \dot{x}_0 \) (say):

\[
\begin{align*}
\dot{x}_3 & \to \dot{x}_2 & \to \dot{x}_1 & \to \dot{x}_0 & \to \dot{x}_{1p} \text{ predicted} \\
& & & & \dot{x}_1
\end{align*}
\]

2) Integrate under fitted curve of \( \dot{x} \) to estimate predicted \( x_{1p} \) at \( t_1 \)

3) Correct derivative at \( t_1 \) via original equation:

\[ \dot{x}_{1c} = f(x_{1p}) \]

Can be iterated to correct more

has many variations

---

*Numerical Recipes* reports this has been superseded by Richardson extrapolation – in particular, Bulirsch-Stoer methods, which extrapolate to zero \( h \) - state-of-the-art?

We'll describe traditional, safe and easy to explain and program - Runge-Kutta.
Second-Order Runge-Kutta (Midpoint) Method

Idea:

1. Use Euler with $h/2$ to find $\hat{x}_{1/2}$
2. Use derivative $f(\hat{x}_{1/2})$ for a full step from $x_0$

Local Error:

$$\hat{x}_1 = x_0 + hf(\hat{x}_{1/2})$$
$$= x_0 + h \left[ f(x_0) + \frac{h}{2} \dot{f}(x_0) + o(h^2) \right]$$
$$= x_0 + hf(x_0) + \frac{h^2}{2} \dot{f}(x_0) + o(h^3)$$

\[\text{first 3 terms of Taylor's series correct}\]

\[\therefore E_{\text{local}} = O(h^3)\]
\[E_{\text{global}} = O(h^2)\]
The 4th Order Runge-Kutta method is a dependable workhouse.
- Classical
- Easy to program

\( f(\cdot) \) evaluated four times per step

\[
\begin{align*}
\dot{x} &= f(t, x) \\
\k_1 &= h f(t_n, x_n) \\
\k_2 &= h f(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}) \\
\k_3 &= h f(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}) \\
\k_4 &= h f(t_n + h, x_n + k_3) \\
x_{n+1} &= x_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)
\end{align*}
\]

Local error \( O(h^5) \)
Global error \( O(h^4) \)

See Num. Recipes also for adaptive step size methods
plus more sophisticated stuff. This will do for Assign 3.
Stability

Simple example:
\[ \dot{x} = -cx \quad \text{one-dimensional} \]

Apply Euler's Method:
\[ x_{n+1} = x_n + h \dot{x}_n = x_n - hcx_n \]
\[ x_{n+1} = (1-ch)x_n \]

Actual solution is \[ x_{n+1} = x_n e^{-ct} = x_n e^{-ch} \]

Converges if \( ch << 1 \), \( h << 1/e \)

In general will be stable if
\[ |1-ch| < 1 \]

otherwise diverges.
\[ h > 2/e \]

In general, for explicit forward methods like Euler, Runge-Kutta, we require
\[ h << 1/\lambda_{max} \] where \( \lambda_{max} = \text{largest eigenvalue} \)

Especially troublesome for "stiff" systems:
Say
\[ \dot{x} = e^{-t} - 1000t \quad \text{solution} \]

\( \lambda_{max} = 1000 \), so we need \( h << 1/1000 \) for stability.
But after a short time, the \( e^{-1000t} \) term is negligible.

Easily leads to
Implicit Methods:

Simple example again.

\[ x_{n+1} = x_n + h \cdot \chi_{n+1} \]  
"Backward Euler"  

\[ = x_n + h(-c \chi_{n+1}) \]

Solve:

\[ \chi_{n+1} = \left( \frac{1}{1+ch} \right) x_n \]

\[ |\chi_{n+1}| = \left| \frac{1}{1+ch} \right| |x_n| \leq |x_n| \]

In this simple case, \( c > 0 \) corresponds to stable first order system, & this is always stable!

Also unconditionally stable for stable, linear, constant coefficient eqns. of any order.

But these are only the simplest cases!

In nonlinear cases, solving the implicit eqn. is hard, and there is no guarantee of stability.

But these techniques are often a big improvement for "stiff" systems — see Num. Recipes for Refinements of Runge-Kutta, etc. to implicit.
Briefly, two-point boundary value problems.

Suppose we are given some conditions at $t=0$ and others at $t=T$ (final). What do we do?

A common approach is the

**Shooting Method:**

guess all conditions at $t=0$ and integrate:

Required boundary guess

\[ \text{use discrepancy to adjust } \]

\[ t=0 \text{ conditions and iterate} \]

desired boundary values

Another method:

**Relaxation:** Start with (hopefully good) guess on grid:

```
```

adjust all values simultaneously

sounds ridiculous, but can be good.

In summary, remember that solving differential equations is an **art** as well as a science.

We have only touched the surface.
Phase-space Picture Gallery

[Gluck87]

Velocity is zero as the pendulum starts its swing. Position is a negative number; the distance to the left of the center.

The two numbers specify a single point in two-dimensional phase space.

Velocity reaches its maximum as the pendulum’s position passes through zero.

Velocity declines again to zero, and then becomes negative to represent leftward motion.

Another way to see a pendulum. One point in phase space (right) contains all the information about the state of a dynamical system at any instant (left). For a simple pendulum, two numbers—velocity and position—are all you need to know.

Making portraits in phase space. Traditional time series (above) and trajectories in phase space (below) are two ways of displaying the same data and gaining a picture of a system’s long-term behavior. The first system (left) converges on a steady state—a point in phase space. The second repeats itself periodically, forming a cyclical orbit. The third repeats itself in a more complex waltz rhythm, a cycle with “period three.” The fourth is chaotic.