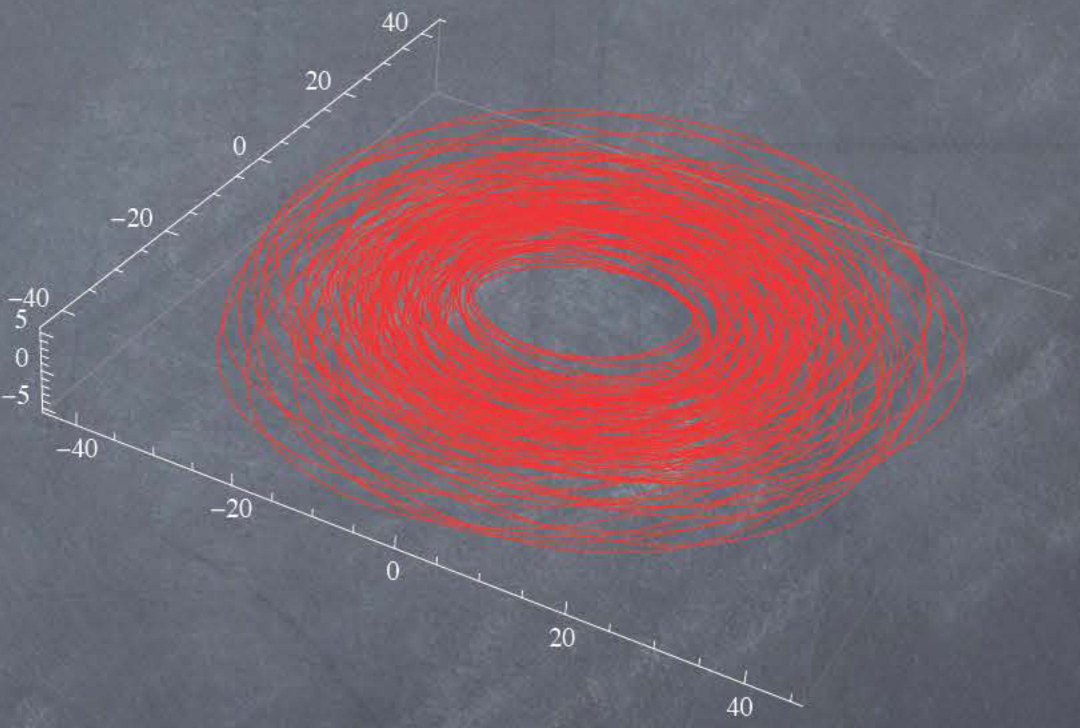
Introduction to Theory and Numerics of Partial Differential Equations I: Introduction and ODEs



Sascha Husa

ICTS Summer School on Numerical Relativity
Bangalore, June 2013

Plan for these lectures

- Lecture 1: **Introduction and ODEs**
  - Basic theory of ODEs
  - solving ODEs with Runge Kutta methods, convergence and error
- Lecture 2: **Mathematical concepts of PDEs**
  - Focus on initial value problems, well posedness and the concept of hyperbolicity
- Lecture 3: **Properties and Stability of Finite Difference Schemes**
- Lecture 4: **Wave equation and Einstein equations in 1+1 dimensions**
- Lecture 5: **Capturing radiation and infinite domains**
- Lab sessions (Python, Fortran 90, Matlab, Mathematica examples):
  - toy model ODEs -> PN binary inspiral -> wave equation -> GR in spherical symmetry.

Goals of these lectures

- Understand the basic problems one faces in solving the Einstein equations as partial differential equations, and the basic ideas of how some of these problems have been solved.
- Be ready to get details from the literature.
- Start to play with some code, be able to solve ODE systems, PDEs in one space dimension.

Lab goals for today

- Write a code than can solve systems of ODEs, using the forward Euler, RK2 and RK4 methods.

$$y' = \lambda y \qquad y'(t) = t^n \qquad y'(t) = sin(t)$$

- Use it on simple ODEs first, check convergence and quantify the numerical error:
- How large can you make the time step? What happens when the time step is too large?
- Does the solution converge to the exact solution?
- Is roundoff error a problem?
- Optional: TOV

Computational infrastructure for NR

- No need to reinvent the wheel - software exists for many of the algorithms/tasks of interest to NR - from specialized libraries to full 3D application suites.
- **Before getting deeper into NR, start with your own home-grown 1D code!**
- Choice of programming language? C, C++, Fortran ≥ 90 for ultimate speed. Basis for current 3D infrastructures.
- Alternatively consider working with Matlab, Python (NumPy, SciPy, ...), Mathematica, or consider to only write numerically intensive Kernels in C, ...
- Learn a general purpose computing environment: e.g. Matlab, Mathematica, Python. Same for data analysis!
- suite of standardized testbeds for NR: www.ApplesWithApples.org
- xAct suite of tensor computer algebra Mathematica packages, http://www.xact.es (J. M. Martin García @ Wolfram)
- Visualisation: Gnuplot, ygraph, VTK programming environment and VTK-based tools such as VisIt highly popular (and free).

Computational infrastructure for NR

- **Cactus Computational Toolkit**: "Framework" for MPI/OpenMP/GPU parallelization, based on user-defined modules called "thorns", since ~1996.
- **Lorene**: LORENE is a set of C++ classes to solve various problems arising in NR, and more generally in computational astrophysics. It provides tools to solve partial differential equations by means of multi-domain spectral methods. [http://www.lorene.obspm.fr/]
- **Einstein Toolkit**, [http://einsteintoolkit.org],
  - collection of open source code for NR, essentially built on Cactus framework.
- **HAD:** open source distributed AMR infrastructure for PDEs [http://had.liu.edu]
- **SpEC** - Spectral Einstein Code [http://www.black-holes.org/] infrastructure for solving PDEs using multi-domain spectral methods. Used by Caltech-Cornell-CITA-Pullman collaboration (SXS), private with many collaborators
- **BAM**: finite difference moving punctures code developed by Jena+, originally started by Brügmann @ AEI, private with many collaborators
- **GR1D** - A New Open-Source Spherically-Symmetric Code for Stellar Collapse to Neutron Stars and Black Holes [http://www.stellarcollapse.org/codes.html]

Solving Einstein's equations

$$G_{ab}[g_{cd}] = R_{ab} - \frac{1}{2} R_c{}^c g_{ab} = 8\pi\kappa T_{ab}[g_{cd}, \phi^A], \qquad R_{bd} = R^a{}_{bad}.$$

$$R^a_{bcd} = \Gamma^a_{bd,c} - \Gamma^a_{bc,d} + \Gamma^m_{bd}\Gamma^a_{mc} - \Gamma^m_{bc}\Gamma^a_{md}, \quad [\nabla_a, \nabla_b]v^c = R^c_{dab} v^d,$$

$$\Gamma^i{}_{k\ell} = \frac{1}{2} g^{im}(g_{mk,\ell} + g_{m\ell,k} - g_{k\ell,m}).$$

- In a coordinate system EEs become set of complicated coupled nonlinear PDEs - need to fix coordinates to fix PDEs, EEs do not correspond to a fixed type of PDE (e.g. hyperbolic).
- All about Einstein's equations -> Baumgarte lecture

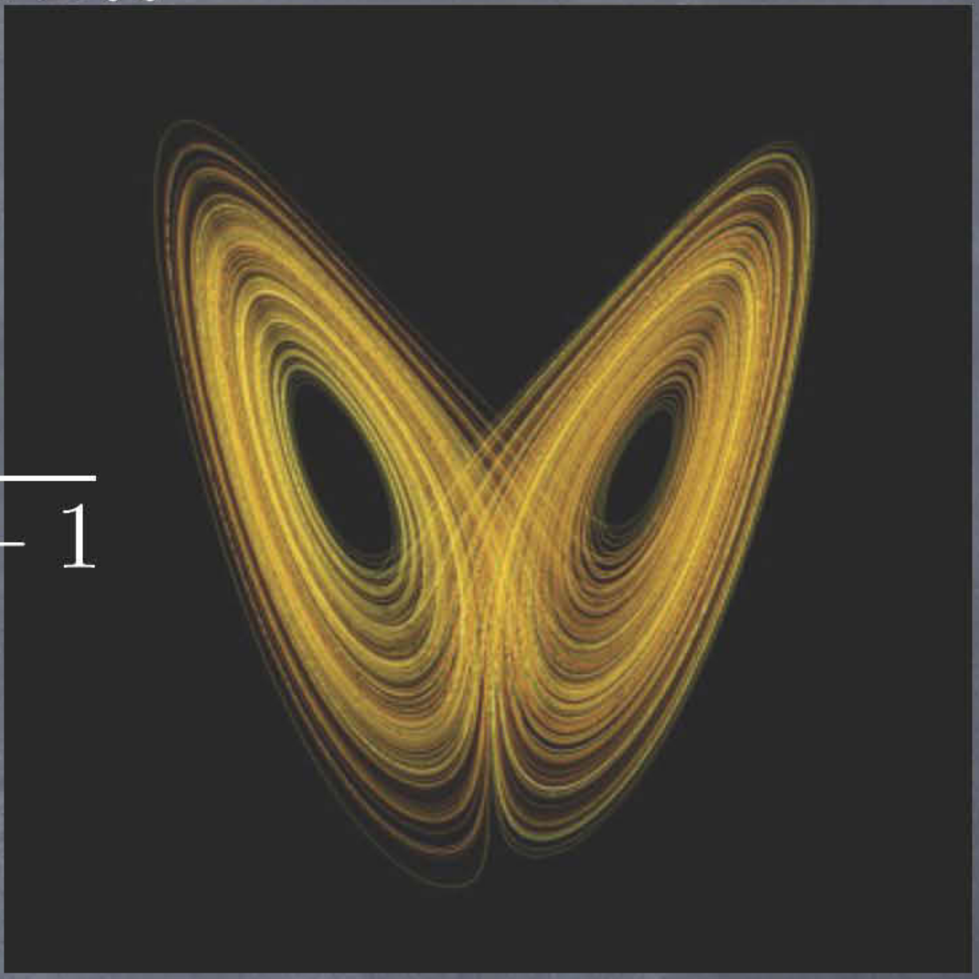
Keys to understand numerics: Conditioning

- Consider model problem F(x,y) = 0
  - How sensitive is the dependence y(x)?
- condition number K: worst possible effect on y when x is perturbed.
- consider perturbed eq. $F(x + \delta x, y + \delta y) = 0$,
- define $K = \sup_{\delta x} \frac{||\delta y||/||y||}{||\delta x||/||x||}$
- K small: well conditioned, K large: ill conditioned,
- K=∞: ill-posed, unstable; K finite: well-posed
- NR: find well-posed PDE problem and for a given problem a gauge that makes K small!

ODEs in a nutshell

- Don't try to understand PDEs without understanding systems of ODEs.
- Can write ODE systems in first order differential form as a "normal form": $y_i'(t) = F_i(t,y_j)$
  - For higher differential order systems, introduce new variables, e.g. $y''(t) = F$: $v := y'$ -> $\{y' = v, v' = F\}$
- Standard result of ODE theory:
  The ODE initial value problem is "well-posed": Given initial data $y_i(t=t_0)$, a unique solution $y_i(t)$ exists at least for some **finite** time $t > t0$.
- A global solution, i.e. for $t \to \infty$ may or may not exist.

ODEs in a nutshell



- For nonlinear ODEs, solutions may blow up in finite time:

$$y' = \lambda y^2, y(0) = y_0 \quad \rightarrow \quad y(t) = \frac{y0}{t\,y0 - 1}$$

- Einstein equations: strong fields -> singularity formation in finite time!
- ODEs may be chaotic in nature, e.g. Lorenz equations (model atmospheric convection, simplified models for lasers, electric circuits, chemical reactions, ...)

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y - x),$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y,$$
$$\frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z.$$

- Lorenz equations are deterministic, but small changes to initial data have a large effect - system is ill conditioned but not ill posed.

ODE boundary value problems

- ODE boundary value problem: e.g. stationary solution in spherical symmetry (singular problem)
- shooting and matching, boundary value problem, eigenvalue problem

Linear systems of ODEs

- consider **constant coefficient** linear ODE systems: for nonlinear equations, we can consider perturbations (can be stable or unstable), coefficients can be considered constant for a short time.

- constant coefficient linear ODE systems can be solved explicitly:

$$y_i' = A_i{}^j y_j \quad \rightarrow \quad y_i(t) = e^{A_i{}^j t} y_j(0)$$

- Compute matrix exponential by transforming A to Jordan form:

$$PAP^{-1} = D + N, \; N^n = 0 \quad \Rightarrow \quad e^{iAkt} = e^{iDkt} e^{iNkt} = e^{iDkt} \sum_{l=0}^{l=n-1} N^l \frac{k^l t^l}{l!}$$

- We can understand the behavior of the solutions in terms of the eigenvalues and eigenvectors of the matrix A.

- Real part of eigenvalues negative: solutions relax to stable steady state.

Numerical Integration of ODEs

- Various techniques are available to obtain exact solutions for certain families/types of ODEs, but general problems, in particular nonlinear ones, have to be solved numerically.
- Consider a simple single ODE: $y'(t) = F(t,y)$
- Replace derivative by a difference expression, e.g.

first order error

$$y'(t) = \frac{y(t+h) - y(t)}{h} - \frac{1}{2}y''(t)h + O(h^2)$$

- Rearrange to obtain the "forward" (explicit) Euler method:

$$y_{n+1} = y_n + h\left[F(t_n, y_n) + \frac{h}{2}y''(t) + O(h^2)\right]$$

- Alternative: backward Euler method - implicit (use e.g. Newton-Raphson to solve equations)

$$y_{n+1} = y_n + hF(t_{n+1}, y_{n+1})$$

Local truncation order

- Error term in the Euler method is first order - we must be able to do better! Use higher order approximations (Taylor)!
- But does Euler actually work? Does the numerical approximation converge? We are only interested in the continuum solution!
- Local truncation order: difference between exact and numerical solution in 1 step:

$$y_{n+1} = R(t_n; y_{n+1}, y_n, \{y_{n-k}\}; h)$$

$$\delta^h_{n+1} = R(t_n; y_{n+1}, y(t_n), y(\{t_{n-k}\}); h) - y(t_{n+1})$$

- The method is consistent if $$\lim_{h \to 0} \frac{\delta^h_{n+1}}{h} = 0$$
- Method is convergent of order p if $\delta^h_{n+1} = O(h^{p+1})$
- Euler methods are consistent and of order 1.

Global truncation order

- Local error is relatively easy to control, but we need to know the global error - the error accumulated in all the steps one needs to reach a fixed time t.
- In the limit h-> 0 we need infinitely many steps, we can suspect that a "bad method" will not let us carry out this limit.
- In an unstable scheme, making a tiny error in each step will diverge in the limit.
- The global error of a p-th order scheme will be $O(h^p)$.

Roundoff error

- Truncation error of a finite difference scheme is not the only source of error on a digital computer!
- We are using numbers with a finite precision, usually we are using double precision numbers as implemented in the machine hardware:

  - Single precision, called "float" in the C language family, and "real" or "real*4" in Fortran. This is a binary format that occupies 32 bits (4 bytes) and its significand has a precision of 24 bits (about 7 decimal digits).
  - Double precision, called "double" in the C language family, and "double precision" or "real*8" in Fortran. This is a binary format that occupies 64 bits (8 bytes) and its significand has a precision of 53 bits (about 16 decimal digits).

- Undefined values: INF or NAN (not a number) - exception handling tends to slow down computations.
- Don't use single prec. unless you really know what you are doing.
- Sometimes quadruple precision comes in handy, expect an order of magnitude slowdown.

Numerical stability of ODEs and stiffness

- Solve a simple linear model equation with Euler's method:

$$y' = \lambda y, y(0) = y_0 \quad \Rightarrow y(t) = y_0\, e^{\lambda t}$$

$$y_{n+1} = y_n + h y_n' = y_n + h\lambda y_n \quad \rightarrow \quad |y_{n+1}|/|y_n| = |1 + h\lambda|$$

- $\lambda > 0$: analytical and numerical solutions grow exponentially.
- $\lambda < 0$: analytical solution decreases exponentially, numerical solution only does this for $h\lambda > -2$ $(h>0)$.
  - For larger time steps the numerical solution exhibits exponential growth, algorithm is unstable!
  - Problem is more serious for ODE systems which exhibit very different decay rates: "stiff"-> very small time steps required.
- [see example codes in Python and Mathematica -> lab session]

Higher order integration schemes

- Basic idea is simple: approximate y′ more accurately, e.g. through a higher order polynomial, compute coefficients with Taylor expansion.

- Standard class of methods: explicit Runge Kutta schemes, **s** stages:

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i,$$

where

$$\begin{aligned}
k_1 &= hf(t_n, y_n), \\
k_2 &= hf(t_n + c_2 h, y_n + a_{21} k_1), \\
k_3 &= hf(t_n + c_3 h, y_n + a_{31} k_1 + a_{32} k_2), \\
&\vdots \\
k_s &= hf(t_n + c_s h, y_n + a_{s1} k_1 + a_{s2} k_2 + \cdots + a_{s,s-1} k_{s-1}).
\end{aligned}$$

- Method is consistent if: $\sum_{j=1}^{i-1} a_{ij} = c_i$ for $i = 2, \ldots, s.$

RK2

- Runge Kutta 2 - "midpoint method"

$$y_{n+1} = y_n + hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hf(t_n, y_n)\right)$$

- Stability: consider $y' = \lambda y$

$$y_{n+1} = Q(h\lambda)y_n$$

- Q(z) is polynomial for RK-methods, for order p:

$$r(z) = e^z + O(z^{p+1})$$

- solution decays (stable) if
  $$|Q(h\lambda)| < 1$$
- "Standard" p-th order RK:

$$Q = \sum_{i=0}^{p} \frac{x^i}{i!}$$

```
def EulerStep(u,t,dt,rhs):
  n=len(u)
  up=np.zeros(n)
  up=u + dt*rhs(u, t)
  return up
```

```
def RK2Step(u,t,dt,rhs):
  n=len(u)
  up=np.zeros(n)
  k1=np.zeros(n)
  k2=np.zeros(n)

  k1 = dt*rhs(u,t)
  k2 = dt*rhs(u + k1, t + dt)

  up = u + 0.5*(k1 + k2)
  return up
```

"Classical Runge-Kutta" - RK4

$$
\begin{aligned}
k_1 &= S(t^{n-1}, f^{n-1}) \\
k_2 &= S\left(t^{n-1} + \frac{\Delta t}{2}, f^{n-1} + \frac{\Delta t}{2} k_1\right) \\
k_3 &= S\left(t^{n-1} + \frac{\Delta t}{2}, f^{n-1} + \frac{\Delta t}{2} k_2\right) \\
k_4 &= S\left(t^{n-1} + \Delta t, f^{n-1} + \Delta t\, k_3\right) \\
f^n &= f^{n-1} + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(\Delta t^5)
\end{aligned}
$$

- Compute max time steps for y'=-y for Euler, RK2, RK4 = 2, 2, 2.785...
- Computational cost/time step = 1,2,4 RHS evaluations.
- For given number of time steps RK4 is the most expensive, for given small global error RK4 is the cheapest.
- In the next lecture we will find out that we can use RK4 for PDEs, but not RK2 or explicit Euler.

Other integration schemes

- Higher order Runge Kutta methods can be constructed, tuned toward efficieny, large time steps, ...
- Runge-Kutta methods are one-step methods. Multistep: reuse information from previous steps (e.g. Adams-Bashforth)
- Efficient solution of many problems requires a variable step size
- Hamiltonian systems (classical mechanics): can exploit properties of such systems and construct integrators to e.g. preserve energy. Geometric integrators (e.g. symplectic integrators) correspond to canonical transformations.

ODE Examples

- Point-particle mechanics
- GR: post-Newtonian approximation of the Einstein equations:
  - expand Einstein equations in powers of v/c - for small velocities this will be an excellent approximation, e.g. solar system, Hulse Taylor binary pulsar, ...
  - In PN, we describe a binary system of e.g. black holes, emitting gravitational waves by a point-particle Hamiltonian and an energy-loss term (GW flux). Point-particle description breaks down before merger.
  - Simplest case: adiabatic inspiral - neglect radial velocity in the source terms.

post-Newtonian black holes

- Start with energy, e.g. as function of separation R or orbital frequency ω: E(R), E(ω). Kepler: $\omega^2 R^3 = G M$.
- PN expansion:

$$\omega^2(R) = \frac{GM}{R^3}\left(1 + f_1(R)\left(\frac{v}{c}\right)^2 + f_2(R)\left(\frac{v}{c}\right)^4 + \dots\right)$$

- Compute energy loss P=-dE/dt to some order in v/c, e.g. at leading order quadrupole formula (see GR text books like Wald)
- To compute the rate of change of any quantity X (e.g. X=ω, R) we write

$$\frac{dX}{dt} = \frac{\frac{dE}{dt}}{\frac{dE}{dX}}$$

post-Newtonian black holes

- To lowest (Newtonian/quadrupole)order:

$$E(R) = m_1 + m_2 - M\frac{\eta}{2}\frac{M}{R}$$

$$E(\omega) = m_1 + m_2 - M\frac{\eta}{2}\left(\frac{(M\omega)^2}{G}\right)^{\frac{1}{3}}$$

$$\frac{dE}{dt} = -\frac{32}{5}\frac{G^4}{c^5}\eta^2\left(\frac{v}{c}\right)^{10}\left(1 + O(v^2) + \dots\right)$$

- Here v is the velocity parameter, η the symmetric mass ratio:

$$v = (GM\omega)^{1/3} \qquad \eta = \frac{m_1 m_2}{(m_1 + m_2)^2}$$

- For GW science, we also need the phase $\frac{d\phi}{dt} = \omega$
- Tomorrow's lab exercise: compute Φ(t), ω(t), R(t) - and error bars!

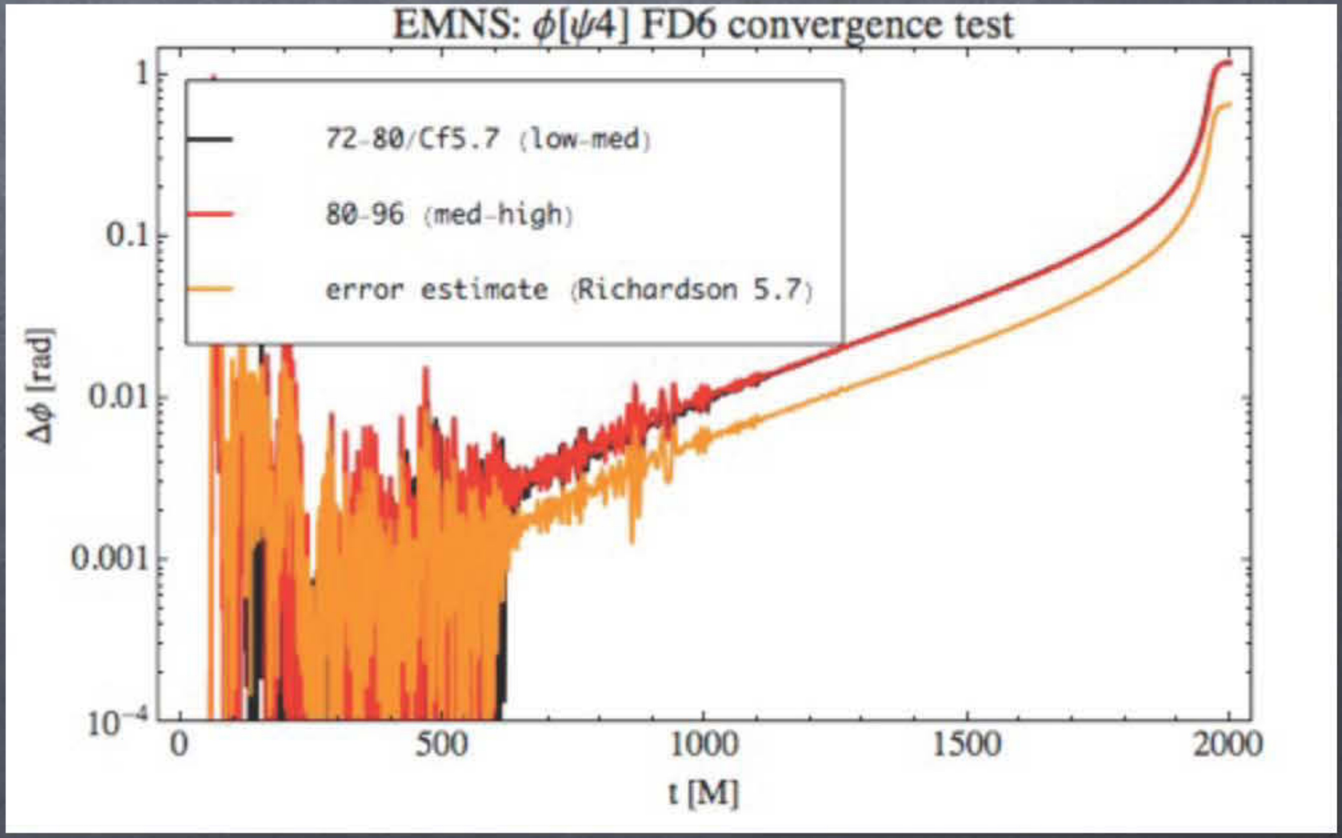
Newton+quadrupole radiation exact solution

$$R(t) = \left(\frac{256}{5}\eta M^3\right)^{\frac{1}{4}} (t_c - t)^{\frac{1}{4}}$$

Convergence example

- We are ultimately only interested in the continuum solution! Is a discretized problem converging to the correct continuum solution? What is the numerical error?
- "... numerical algorithms can be considered as discrete dynamical systems around critical points. (equilibria)." [internet pick, http://www2.de.unifi.it/anum/trigiante/rodid.pdf]
- convergence:

$$X(\Delta x) = X_0 + e\Delta x^n + O(\Delta x^{n+1})$$



  - 3 resolutions determine X_0, e, n
  - consistency: check n
  - then compute X_0

Convergence example

- e.g. choose $\Delta x = h, h/2, h/4$.

$$X(\Delta x) = X_0 + e\Delta x^n + O(\Delta x^{n+1})$$

- derive:

$$\frac{X(h) - X(h/2)}{X(h/2) - X(h/4)} = \frac{h^n - \left(\frac{h}{2}\right)^n}{\left(\frac{h}{2}\right)^n - \left(\frac{h}{4}\right)^n} = 2^n$$

- check that ratio of differences approximates 2^n
- The better the resolution, the better the theoretical ratio should be approximated.
- 2 reasons for why that may not work:
  - algorithm is not what you think it is - converges at different order
  - h not yet small enough