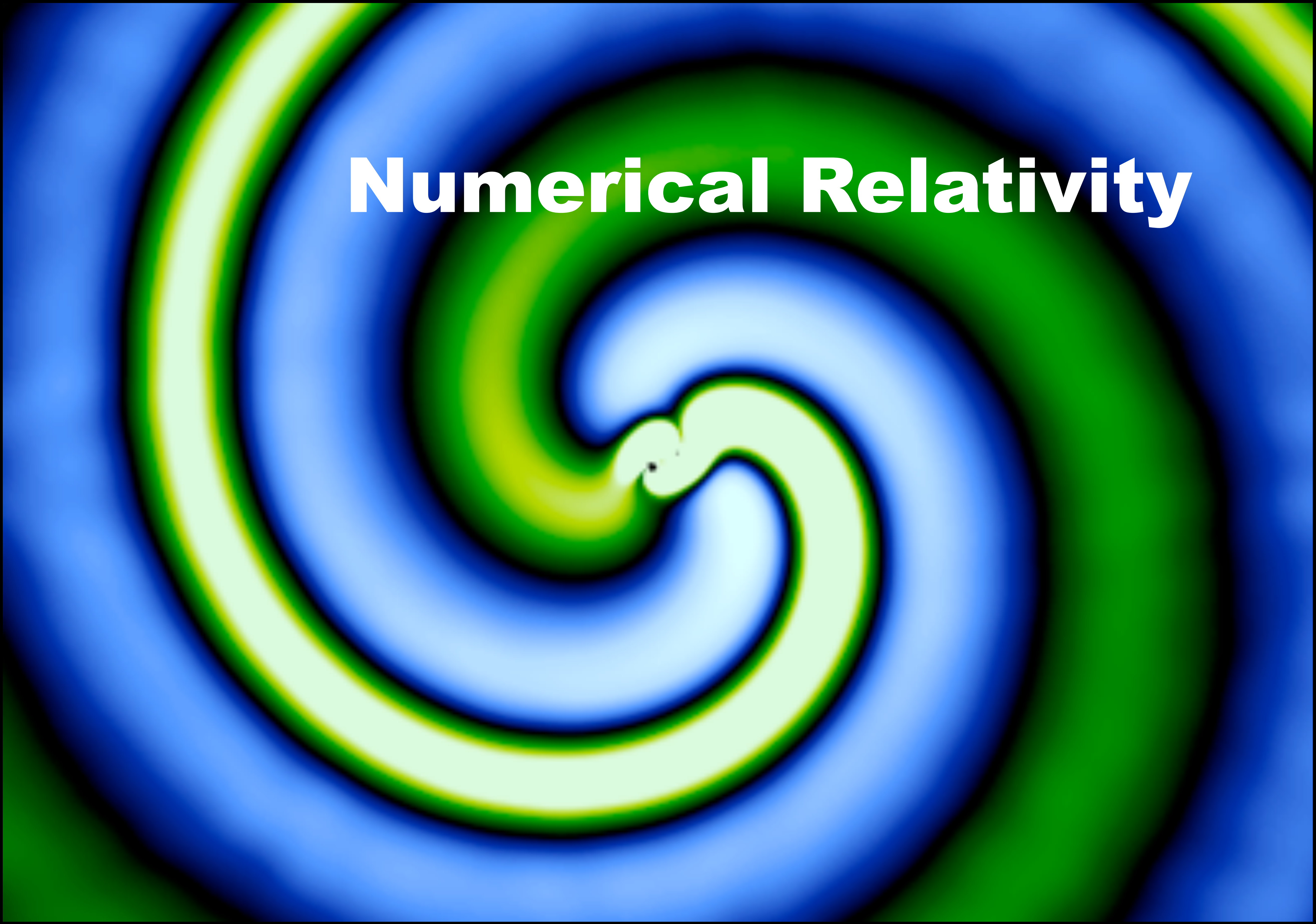
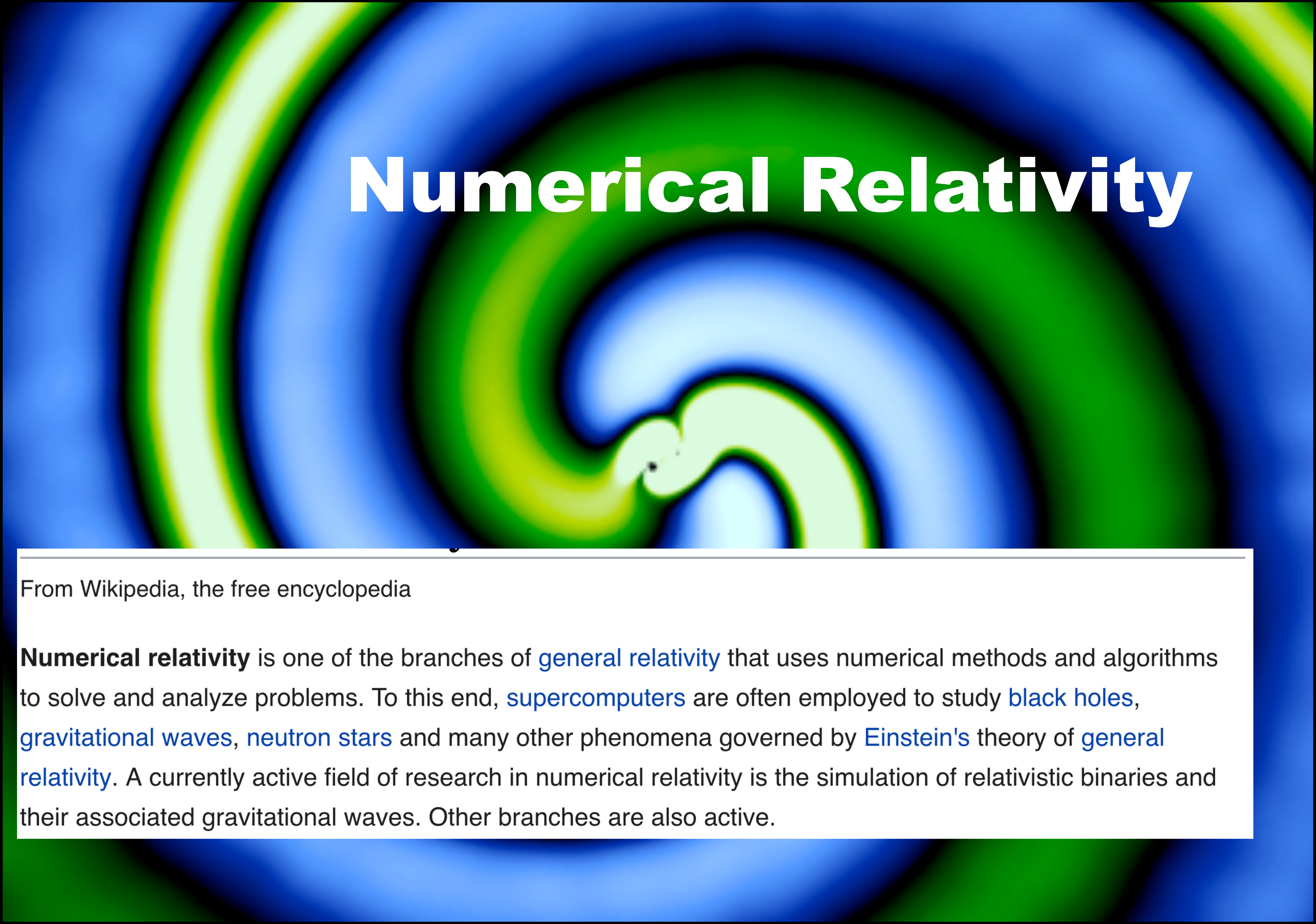
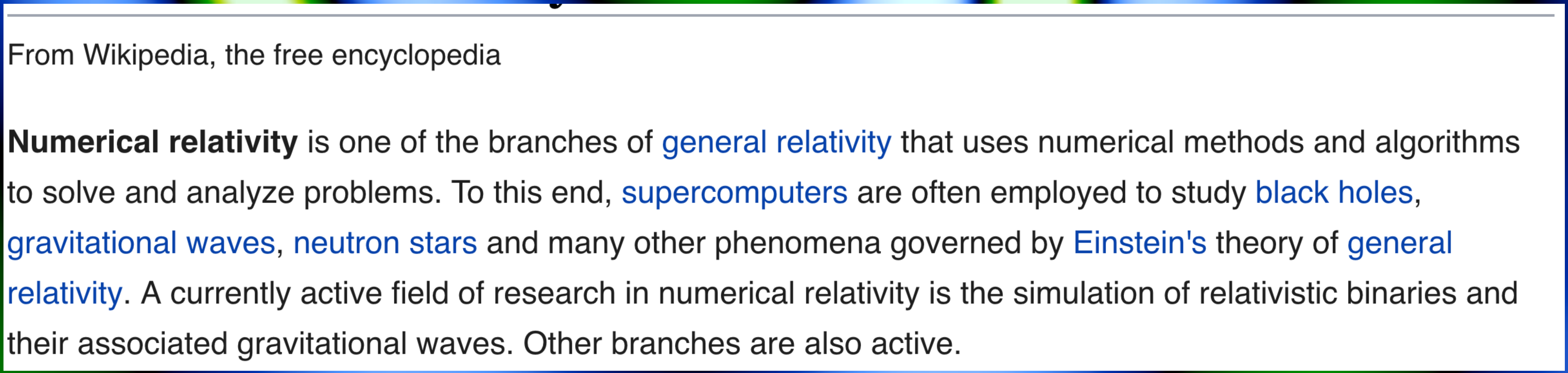
Numerical Relativity



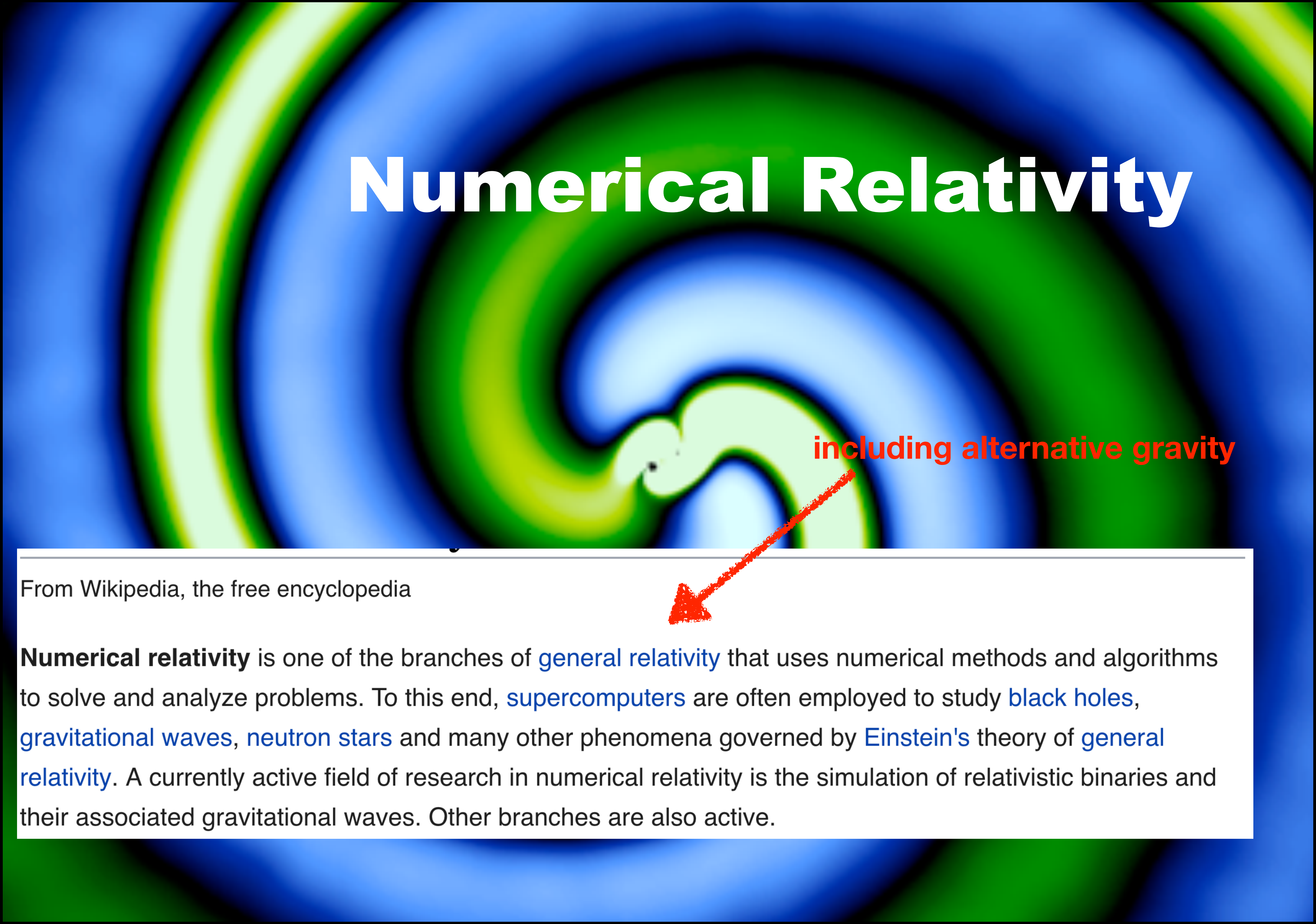
Numerical Relativity





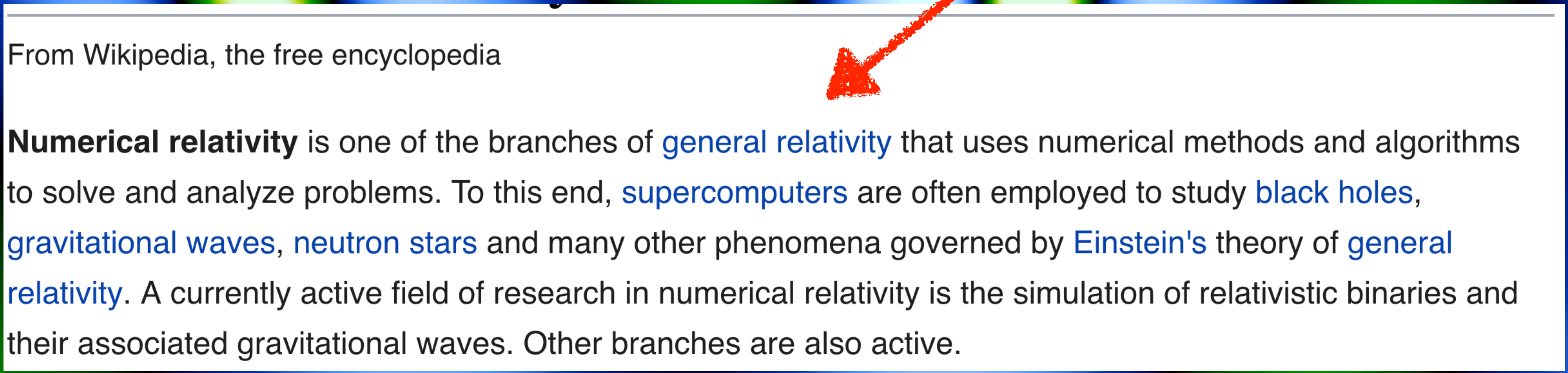
From Wikipedia, the free encyclopedia

Numerical relativity is one of the branches of general relativity that uses numerical methods and algorithms to solve and analyze problems. To this end, supercomputers are often employed to study black holes, gravitational waves, neutron stars and many other phenomena governed by Einstein's theory of general relativity. A currently active field of research in numerical relativity is the simulation of relativistic binaries and their associated gravitational waves. Other branches are also active.



Numerical Relativity

including alternative gravity



From Wikipedia, the free encyclopedia

Numerical relativity is one of the branches of general relativity that uses numerical methods and algorithms to solve and analyze problems. To this end, supercomputers are often employed to study black holes, gravitational waves, neutron stars and many other phenomena governed by Einstein's theory of general relativity. A currently active field of research in numerical relativity is the simulation of relativistic binaries and their associated gravitational waves. Other branches are also active.

- Mathematical problems and exact solutions have dominated GR until recently.

  Deep insights gained: positive mass theorem, nonlinear stability of Minkowski spacetime . . .

- Astrophysics, cosmology, general understanding of the solution space of the EE require approximate solutions – analytical and numerical!

- Numerical solutions allow to study the equations (in principle) without simplifying physical assumptions, and allow mathematical control over the convergence of the approximation!

What we will talk about, and what not.

- Only talk about classical gravity, no computational quantum gravity!
- Solve Einstein Equations as PDEs, alternatively discretize geometry directly (e.g. Regge calculus, discrete differential forms, . . .).

10 lectures + some practical problems

Practical Problems

- Choose 1 of 3 Tracks:
  - ODEs: post-Newton black holes to leading order.
  - Wave equation in 1+1 dimensions.
  - Scalar field in AdS
- Choose a programming language you are familiar with. I can help with Fortran, C, Python, Mathematica.
- Some worked out codes can be provided, but try yourself first.

1. Initial value problems for GR

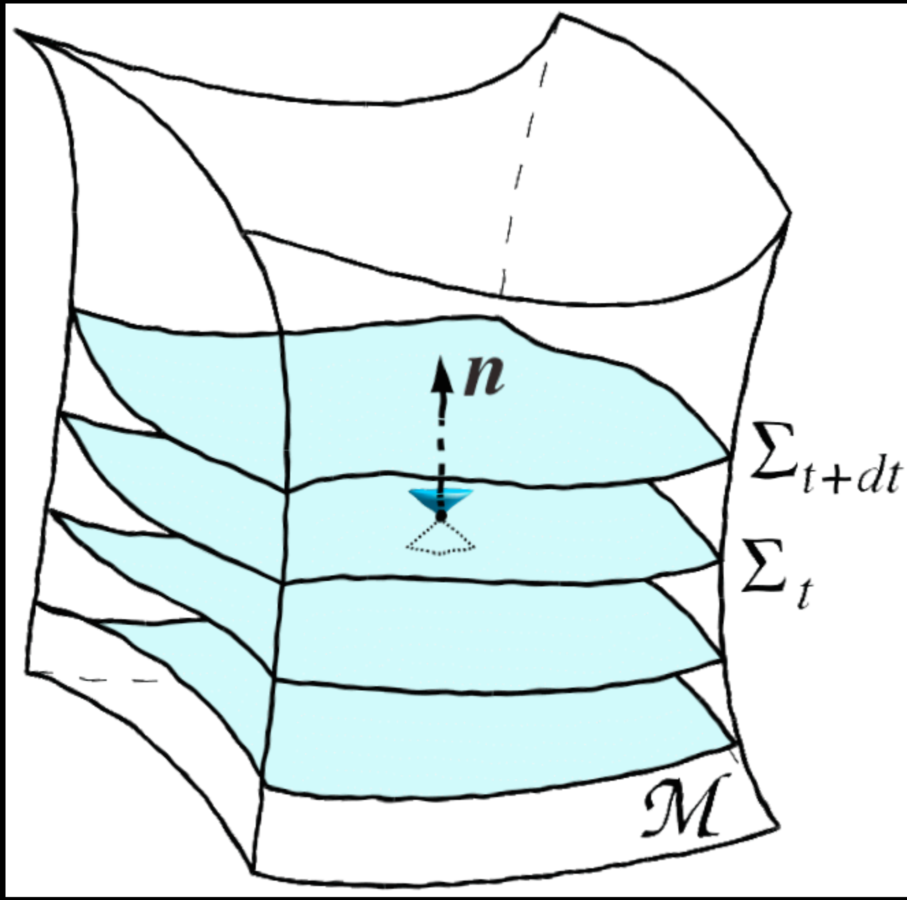
Motivation

- Classical physics is formulated in terms of PDEs for tensor fields.
- To understand a physical theory (GR, Maxwell, QCD, ...) requires to understand the space of solutions of the PDEs that describe it.
  - What predictions to these solutions make for observations?
- Need a systematic=algorithmic way to find approximate solutions of PDEs: perturbation approaches, numerical analysis.
- EEs are intrinsically 4-dimensional
  How/where should we specify boundary conditions = select the physical solution?
  - First sort out what can be chosen and what is then determined by equations !
  - Time evolution problems: given initial data, does a unique time evolution exist? Does it depend continuously on the initial data? Predictability! Important source of physical intuition!

Q: Does the theory have an initial value formulation? Yes! Many!

Initial value problems

$$G_{ab} = \frac{8\pi G}{c^4} T_{ab}$$ -— starting 1950's -—>



Choose coordinates for spacetime =>
~ 10 coupled nonlinear wave eqs., complex source terms.

- The known fundamental theories of nature (GR, elektro-weak theory, QCD (Yang-Mills)) are gauge theories, the presence of gauge freedom leads to constraints - restrictions on the space of possible initial data for a time evolution problems, which typically take the form of elliptic boundary value problems.

  Preserving constraints numerically well understood for E&M, not GR.

Types of PDEs (linear for now)

- Can classify by the type of "problem" that can naturally be associated with a PDE: initial/initial boundary // boundary value problems.

- Standard types:

  - **hyperbolic**, generalize wave equation: information propagates with finite speed
  $$u(\vec{x},t)_{,tt} = \Delta u(\vec{x},t)$$

  - parabolic: generalize heat equation, well posed only forward in time, information propagates instantaneously
  $$u(\vec{x},t)_{,t} = \Delta u(\vec{x},t)$$

  - Schrödinger equation: information propagates instantaneously
  $$u(\vec{x},t)_{,t} = i\Delta u(\vec{x},t)$$

  - **elliptic**, e.g. Laplace equation:
  $$\Delta u(\vec{x}) = 0$$

Well-posedness and stability for evolution equations

Continuum problem:

- WP: A unique solution exists (when gauge is chosen), depends continuously on initial data. Can formulate continuity as

$$\exists K, a \in R : \quad ||u(t)|| \leq K e^{a\,t} ||u(0)|| \quad \forall u(0)$$

- Exponential growth (instability) ok, arbitrarily fast growth not.

  - “mode stability”: can’t have modes which grow arbitrarily fast

  - typical ill-posed problems: Higher frequencies correspond to larger a, K -> better resolution, worse solution.

Discrete problem:

- **WP (stable) in numerical context for iterative problem ($e^{\lambda t}$ ok, $e^{\lambda n}$ not)**

$$v^{n+1} = Q(t_n, v^n, v^{n+1}) v^n : \quad ||v^n|| \leq K e^{\alpha t_n} ||v^0|| \quad \forall v^0$$

- **Lax equivalence theorem: “a consistent (formally convergent) finite difference scheme for a linear PDE for which the initial value problem is well posed is convergent iff it is stable.''**

Key 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 + δx, y + δy) = 0,
- define

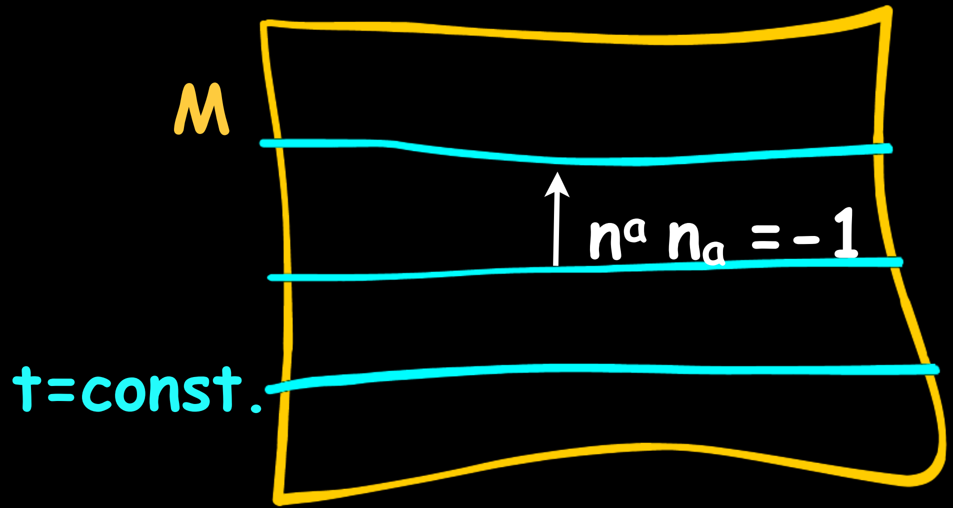
$$K = \sup_{\delta x} \frac{||\delta y||/||y||}{||\delta x||/||x||}$$

- 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!

Initial Value formulation of a simple gauge theory: Maxwell

- 4-dimensional formulation: $\nabla_{[a}F_{bc]} = 0, \quad \nabla_b F^{ab} = j^a$

- **3+1 decomposition:** Introduce a space-time split, define hypersurfaces of constant time by time-like unit normal n^a, electric + magnetic fields E^a, B^a.



$$E^a = F_{ab} n^b, \qquad B^c = \frac{1}{2} F_{ab}\, {}^3\epsilon^{abc}$$

- Get 2 evolution equations (contain time derivs.), in flat space:

$$\partial_t E^a = \epsilon_{abc} \partial^b B^c - 4\pi j_a, \qquad \partial_t B^a = -\epsilon_{abc} \partial^b E^c$$

- Get 2 constraint equations (contain no time derivs.):

- Maxwell equations need to be solved consistently with equations for j^a, ρ $\quad \partial_a E^a = 4\pi\rho, \qquad \partial_a B^a = 0$

Maxwell II

- **Exercise:** show that constraints propagate (always satisfied by virtue of the evolution equations, if satisfied at t=0)

- Initial value problem makes sense: constraints are preserved, fpr given initial data a unique time evolution exists, which depends continuously on initial data = **well-posed initial value problem**

- Information propagates at the speed of light. We will soon understand connection between propagation speeds and the property of an IVP to be well-posed!

Maxwell III

$$F_{ab} = \nabla_a A_b - \nabla_b A_a \;\Rightarrow\; \nabla^a \left(\nabla_a A_b - \nabla_b A_a\right) = j_b$$

- Using vector potential A additional gauge issues appear!
  Lorentz gauge -> Wave equation: $\nabla^a A_a = 0 \;\Rightarrow\; \nabla^a \nabla_a A_b = j_b$

- Numerical ED is difficult (preserve constraints!), but well understood: analytical formulation, numerical algorithms, comparison with experiment!

- curved background:

$$\mathcal{L}_n D_i E^i = -K D_i E^i, \qquad \mathcal{L}_n D_i B^i = -K D_i B^i$$

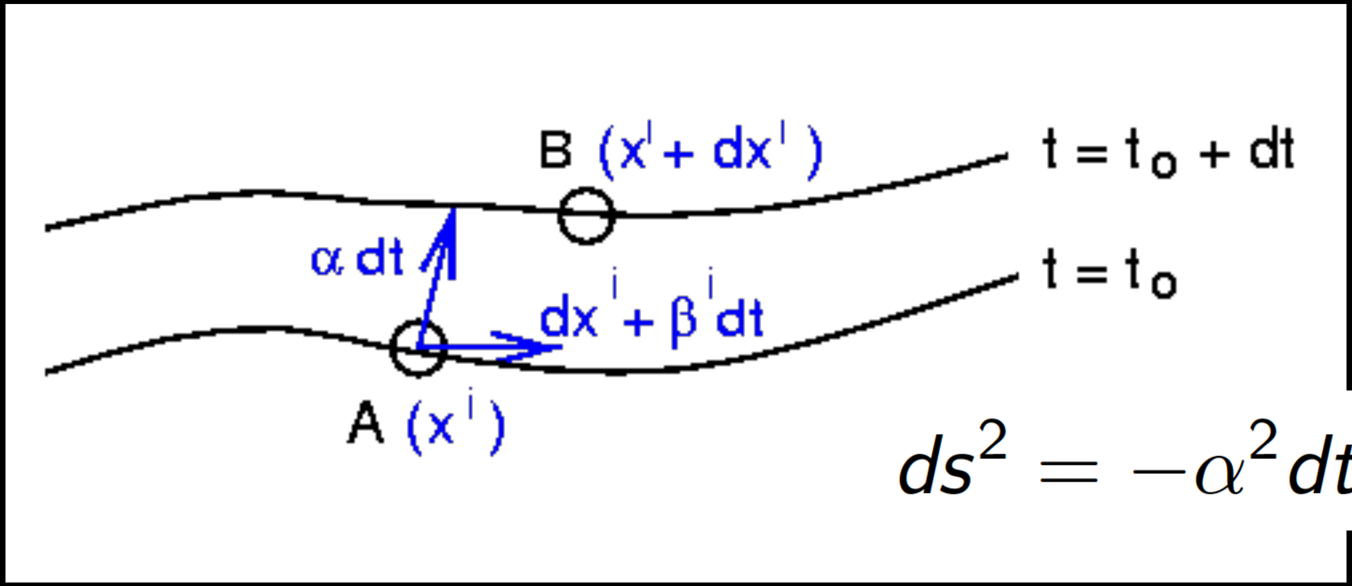
- In collapsing case (K < 0)) instability of constraints!

  - Well-posedness is necessary but not sufficient to accurately approximate the continuum problem with finite precision!

- Solution for Maxwell: use $\sqrt{g} E^a, \sqrt{g} B^a$. GR ?

Fast track 3+1 decomposition for GR

Simplest way to get PDEs from the Einstein Equations:



Chose coordinates $\{x^i , t\}$
(i = 1, 2, 3),
then "read off" metric in the form:

$$ds^2 = -\alpha^2 dt^2 + h_{ab}(dx^a + \beta^a dt)(dx^b + \beta^b dt),$$

hab is a positive definite matrix (Riemannian metric on the 3-spaces of constant time), and α= 0.

4 functions α, β^i are freely specifiable, steer coordinate system through spacetime as time evolution proceeds – physical result is independent of this choice (diffeomorphism invariance).

PDEs resulting from ansatz are of 2nd order for h, split into 2 parts: 4 constraints (no second time derivatives) & 6 evolution equations.

Projections and the Induced Metric

Given foliation: write all tensors in terms of “horizontal” and “vertical” parts!

Let $n_a = -\alpha \nabla_a t$ denote the future timelike unit normal to Σ & define

$$N^a{}_b := -n^a n_b, \qquad h^a{}_b := \delta^a{}_b - N^a{}_b,$$

check they are in fact the desired projection operators (exercise!):

$$h^a{}_b h^b{}_c = h^a{}_c, \qquad N^a{}_b N^b{}_c = N^a{}_c, \qquad h^a{}_b N^b{}_c = 0,$$

$h^a{}_b$ projects onto the tangential, and $N^a{}_b$ onto the normal directions,

$$h^a{}_b n^b = 0, \qquad N^a{}_b n^b = n^a.$$

Apply first to metric $\rightarrow$ induces a tensor field h_{ab} by

$$h_{ab} = g_{ab} + n_a n_b = g_{cd} h^c{}_a h^d{}_b.$$

h_{ab} is purely horizontal, positive definite and nondegenerate for horizontal vector fields (exercise!) $\rightarrow$ natural Riemannian metric on Σ.

Induced Curvatures

2 curvatures associated with embedding of Σ in M:

- intrinsic: Riemann tensor
- extrinsic: describes how Σ bends in M.

Natural derivative operator D_a associated with h_{ab}:

$$D_c T^{a_1 \dots a_r}_{b_1 \dots b_s} := h^{c}_{c'} h^{a_1}_{a'_1} \dots h^{a_r}_{a'_r} h^{b'_1}_{b_1} \dots h^{b'_s}_{b_s} \nabla_{c'} T^{a'_1 \dots a'_r}_{b'_1 \dots b'_s} \tag{2}$$

$\rightarrow$ define Riemann tensor of ${}^3R_{abcd}[h_{ef}]$.

Extrinsic curvature:

$$K_{ab} := h^c{}_a h^d{}_b \nabla_c n_d = \frac{1}{2} L_n h_{ab} = \text{“velocity”},$$

Relation of the intrinsic and extrinsic curvatures of Σ to the curvature of M is given by two crucial geometric identities, the Gauss-Codazzi Eqs.:

$${}^3R_{abc}{}^d = h_a{}^{a'} h_b{}^{b'} h_c{}^{c'} h_{d'}{}^d R_{a'b'c'}{}^{d'} - K_{ac} K_b{}^d + K_{bc} K_a{}^d, \tag{3}$$

$$D_a K^a{}_b - D_b K^a{}_a = R_{cd} n^d h^c{}_b. \tag{4}$$

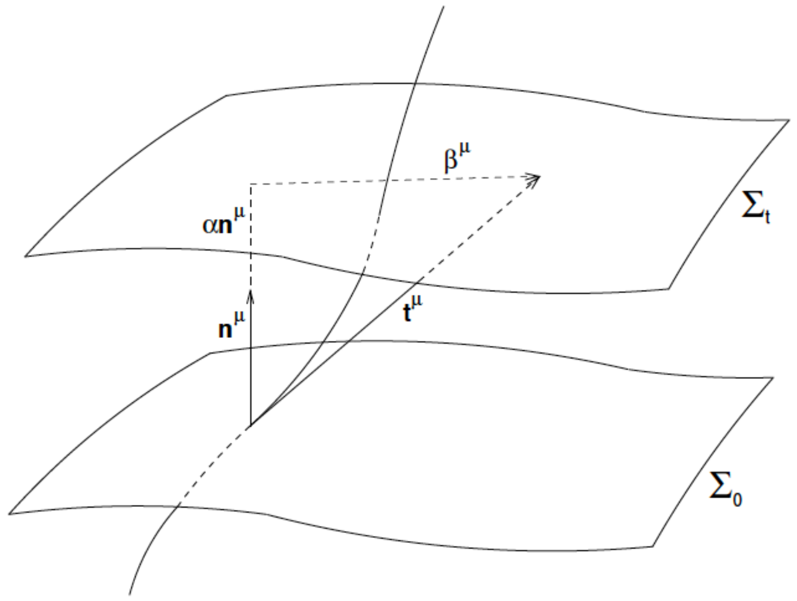
Threadings of spacetime

Consider changes of tensors $T^{\cdots}_{\cdots}$ along the integral curves of a *time flow vector* t^a, given by Lie derivative:

$$\dot{T}^{\cdots}_{\cdots} := \mathcal{L}_t T^{\cdots}_{\cdots} = \frac{\partial}{\partial t} T(x^\alpha, t) \quad \text{in adapted coordinates} \{x^\alpha, t\}.$$

Spacetime engineering: "threading" t^a dynamically "steers" spacetime evolution. Decompose t^a into a normal and a tangential component:

$$t^a = \alpha n^a + \beta^a, \quad \beta^a n_a = 0, \qquad t^a \text{ timelike if } \beta^a\beta_a - \alpha^2 < 0. \tag{5}$$



"Lapse" α determines "how fast time elapses".

"Shift vector" β^a shifts spatial coordinate points with time evolution.

Lapse and shift are not determined by EEs $\Rightarrow$ plenty of rope to shoot yourself in the foot!

Projecting $G_{ab} = 0$

Now use EEs – compute all projections of $G_{ab} = \kappa T_{ab}$ using Gauss-Codazzi (3,4)!

Projections with n^a yield:

$$0 = |G_{bc} n^c h^b{}_a = h^b{}_a R_{bc} = D_a K^a{}_b - D_b K^a{}_a \tag{6}$$

$$0 = G_{ab} n^a n^b = \frac{1}{2}\left({}^3R + (K^a{}_a)^2 - K_{ab}K^{ab}\right) \tag{7}$$

No time derivatives of K_{ab} – they are relations between the initial data h and K, which cannot be freely specified – the constraint equations of GR!

$G_{ab} h_c{}^a h_d{}^b$ yields evolution equation:

$$\begin{aligned} \dot{K}_{ab} &= -D_a D_b \alpha + \beta^c D_c K_{ab} + K_{cb} D_a \beta^c - K_{bc} D_a \beta^c \\ &+ \alpha\left({}^3R_{ab} + K_c{}^c K_{ab}\right) \end{aligned} \tag{8}$$

WRITE OUT hdot equation!!!!!!

Bianchi-Id. $(\nabla^a G_{ab} = 0) \Rightarrow$ Constraints propagate!
(What happens for small initial violations?)

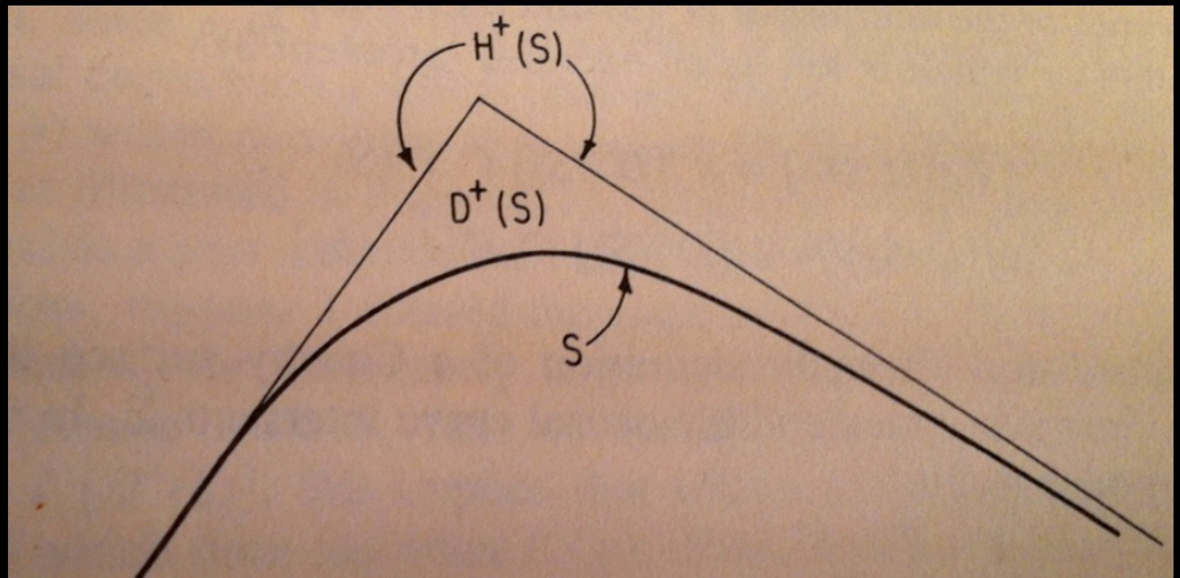
Mid 70's: let's just code them up and collide black holes!

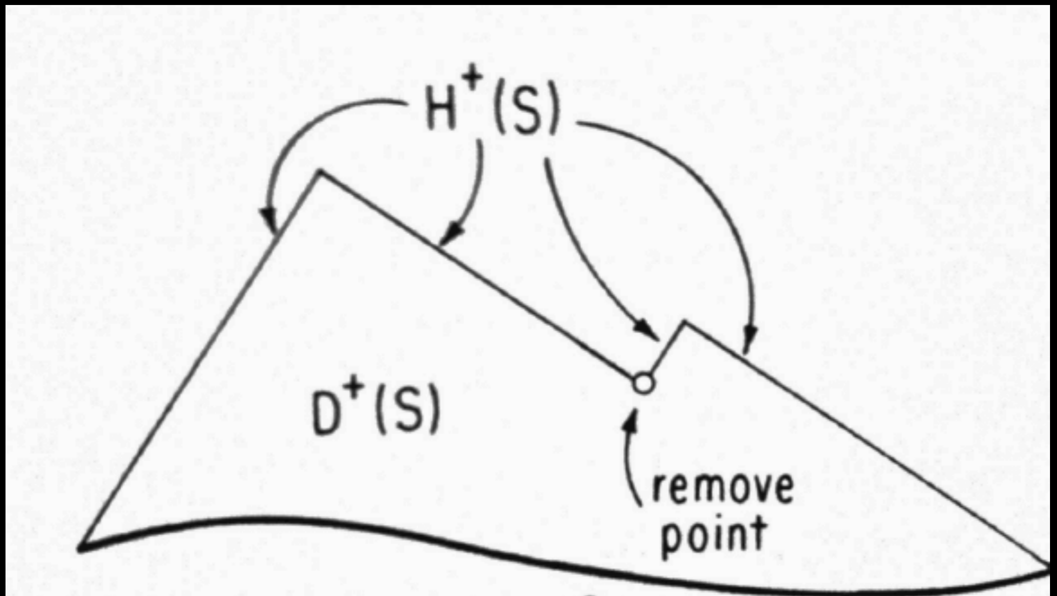
Domain of dependence

- Let S be a 3-D "hypersurface of constant time" [an achronal (non-timelike) embedded submanifold of a manifold M (points of S can not communicate causally].
- Future domain of dependence $D^+(S)$ [analogous for $D^-(S)$]:

$$D^+(S) = \left\{ p \in M \middle| \begin{array}{l} \text{every past inextendible causal curve;} \\ \text{through } p \text{ intersects } S. \end{array} \right.$$

- If nothing can travel faster than light, any signal sent to $p \in D^+(S)$ must have registered on S. Thus, given initial conditions on S, we should be able to predict what happens at p.





Global hyperbolicity

- $D(S) = D^+(S) \cup D^-(S)$
- A set such that $D(\Sigma) = M$ is called a Cauchy hypersurface, is a snapshot of the universe a spacetime which possesses a Cauchy hypersurface is called globally hyperbolic.
- Theorem (see e.g. Wald, chapter 8):

  Let (M, g_{ab}) be a globally hyperbolic spacetime. Then (M, g_{ab}) allows a global time function t, such that each surface of constant t is a Cauchy surface, and the topology of M is $R \times \Sigma$, where Σ denotes any Cauchy surface.
- Globally hyperbolic spacetimes are those which can be constructed as an initial value problem.
- Globally hyperbolic spacetimes do not allow closed timelike curves (time machines).
- Spacetimes with time machines are not “predictable”.

Beyond the Cauchy Problem

- Explain alternatives to Cauchy problem on blackboard:
  - characteristic initial value problem
  - Hyperboloidal initial value problem

2. ODEs

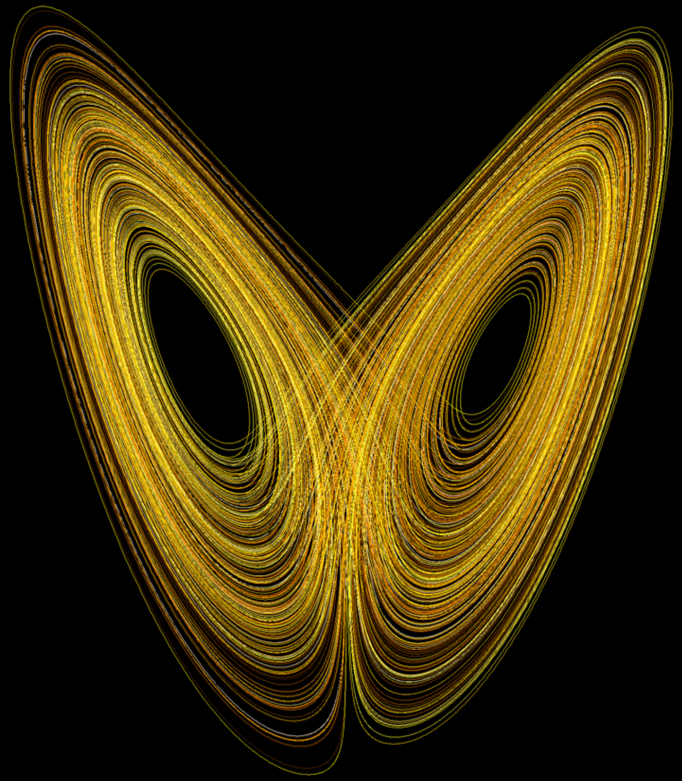
ODEs in a nutshell

- 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.

Nonlinear ODEs

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

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

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 behaviour 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

$$y'(t) = \frac{y(t+h) - y(t)}{h} - \underbrace{\frac{1}{2}y''(t)h}_{\text{first order error}} + O(h^2)$$

- 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.

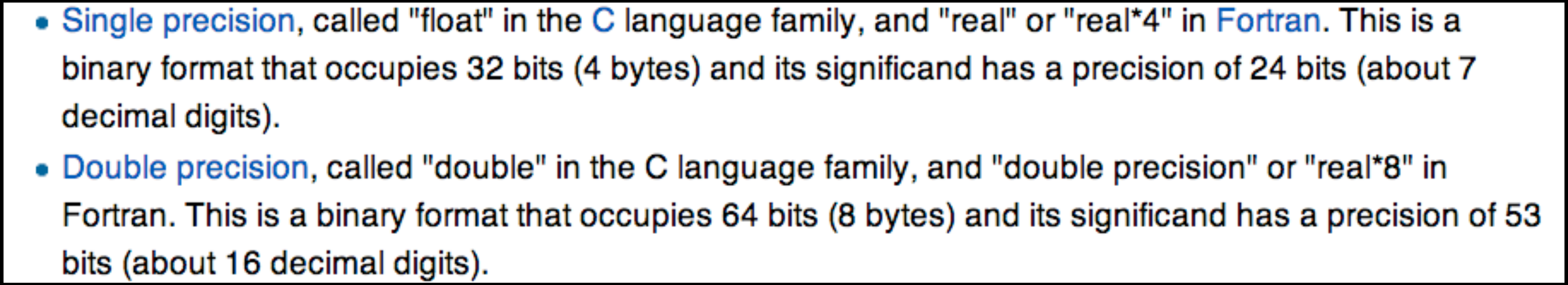
Global truncation order

- 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 & stiffness

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

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

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

- λ > 0: analytical and numerical solutions grow exponentially.

- λ < 0: analytical solution decreases exponentially, numerical solution only does this for hλ > -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:

$$\sum_{j=1}^{i-1} a_{ij} = c_i \text{ for } i = 2, \ldots, s.$$

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’ = λ y

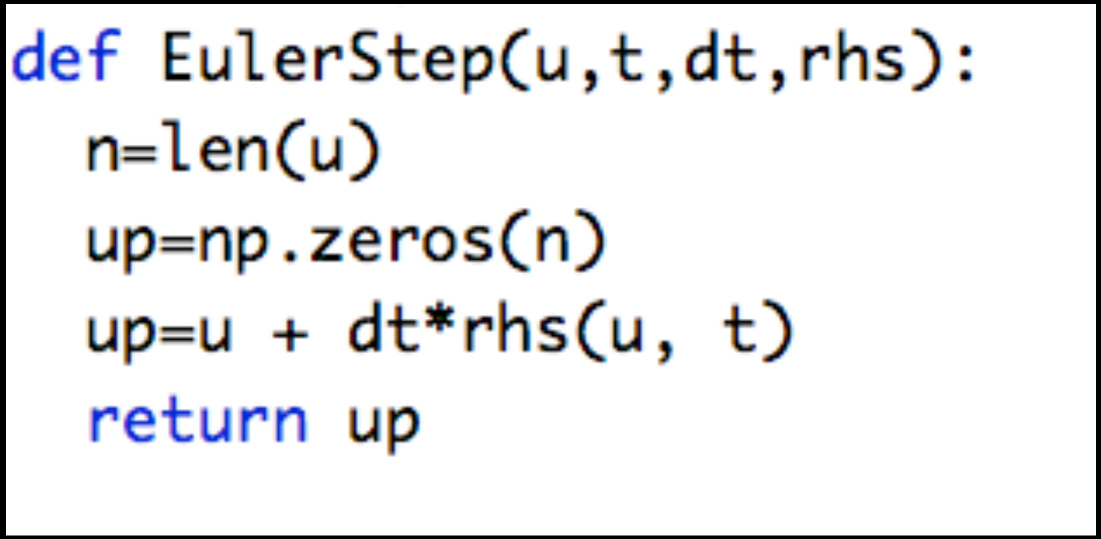
$$y_{n+1} = Q(h\lambda)y_n$$

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

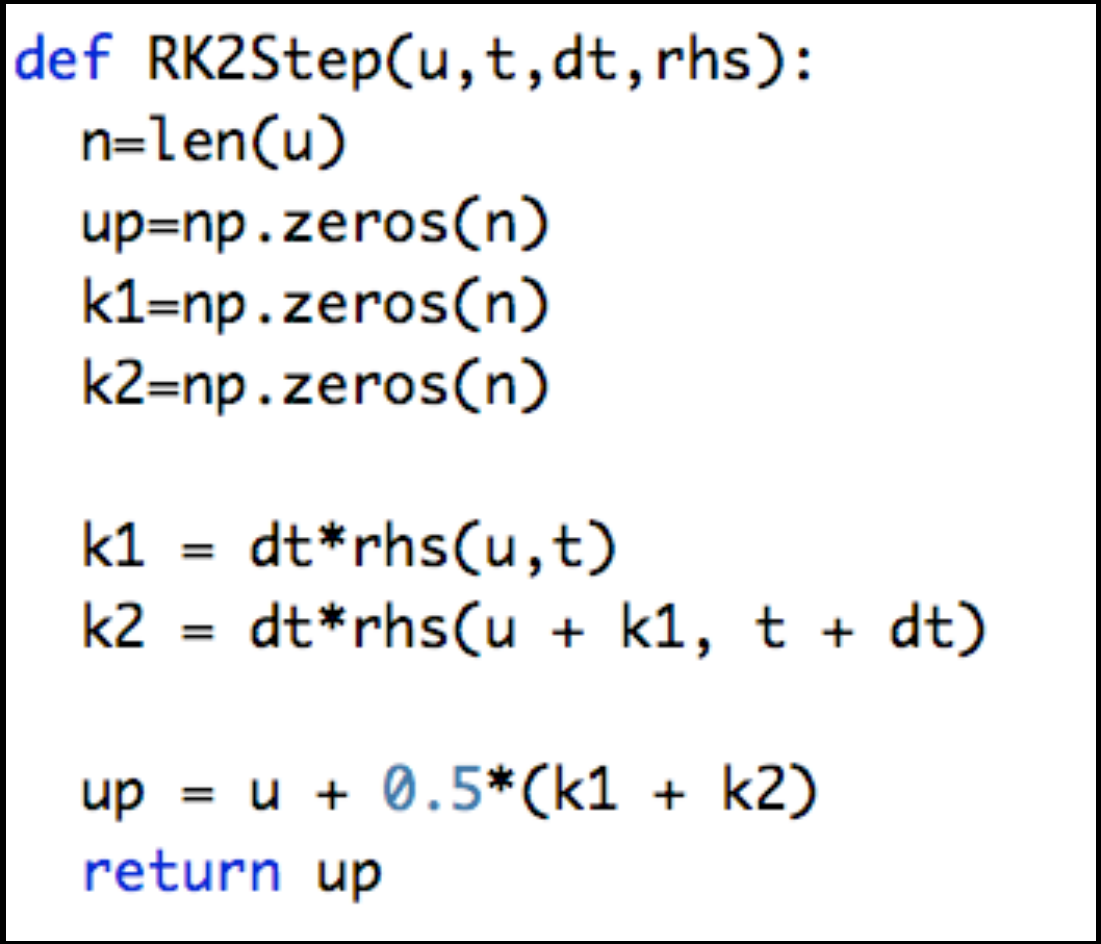
$$r(z) = e^z + O(z^{p+1})$$

- 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.

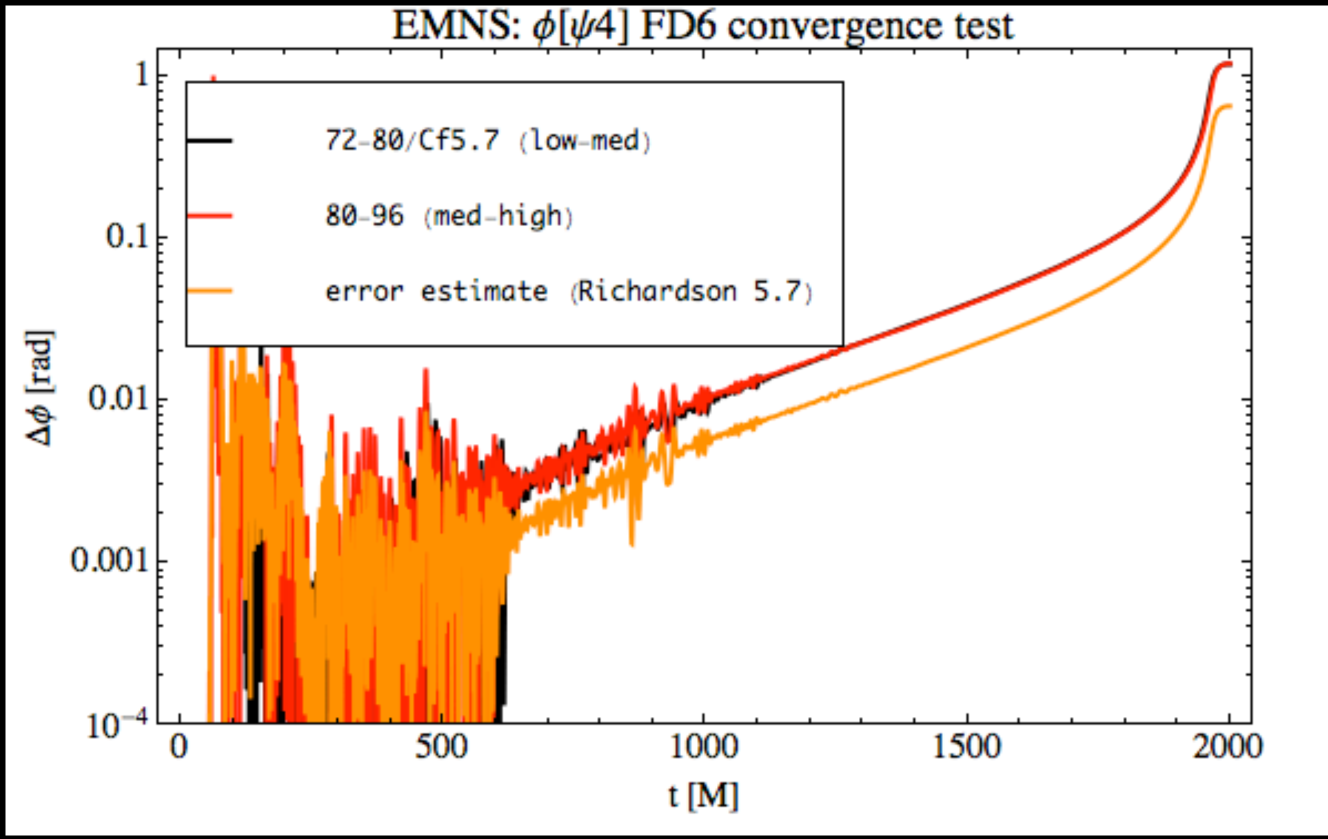
Convergence

- We are ultimately only interested in the continuum solution! Is a discretized problem converging to the correct continuum solution? What is the numerical error?

  - 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 Δx = 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