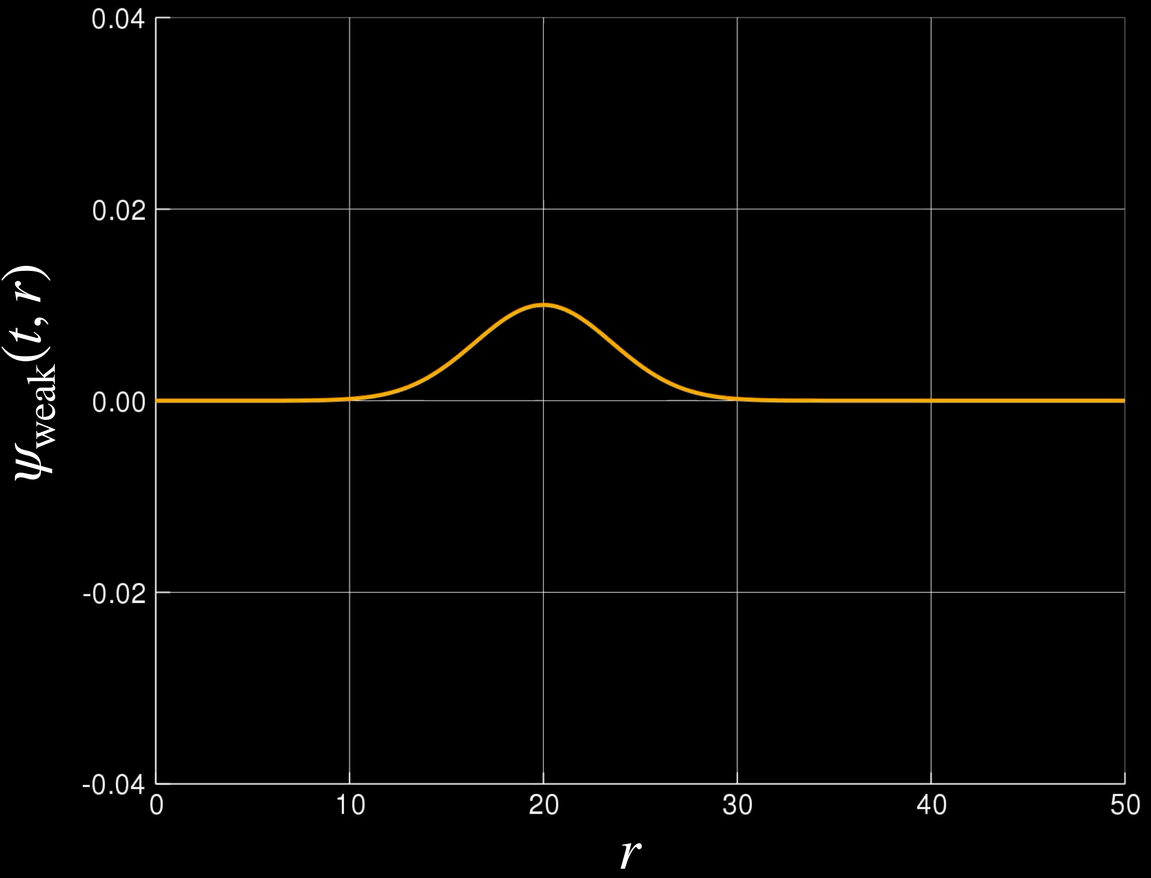
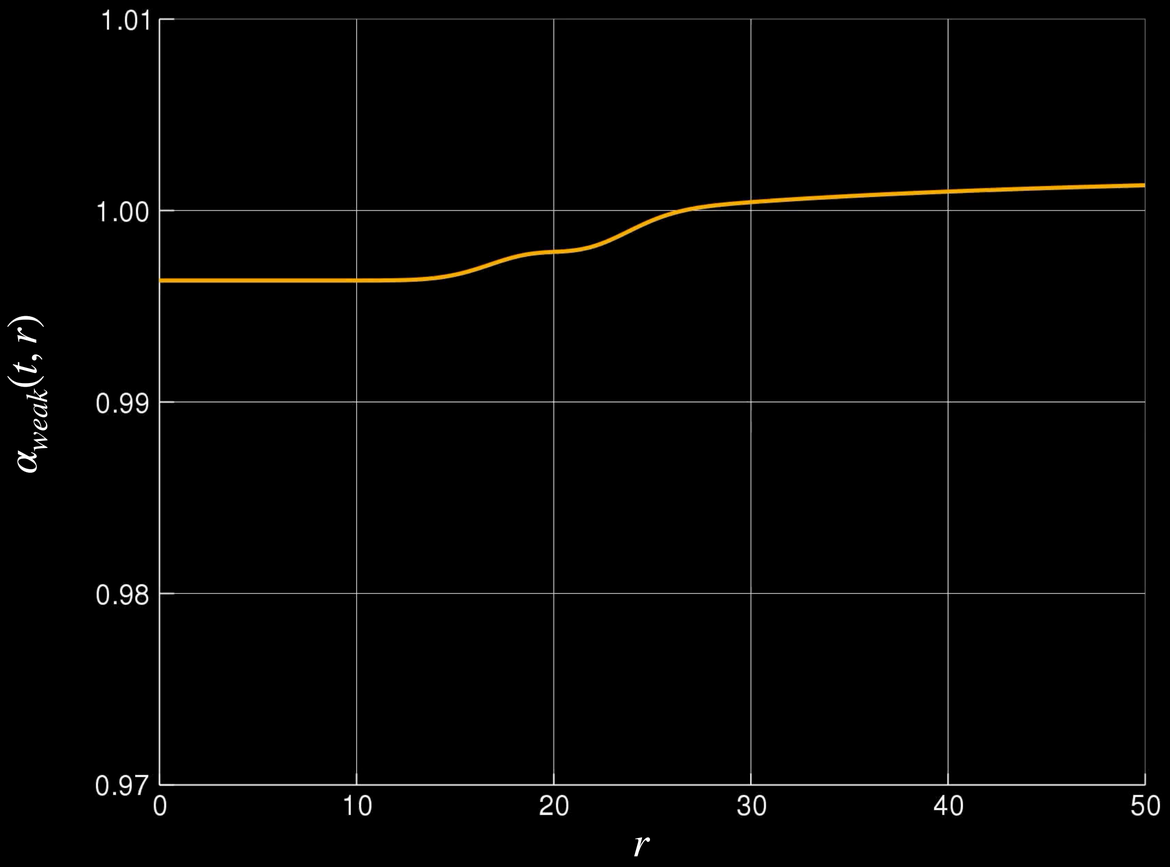
New, User-Friendly Codes to Study Critical Collapse

Dr. Leonardo R. Werneck

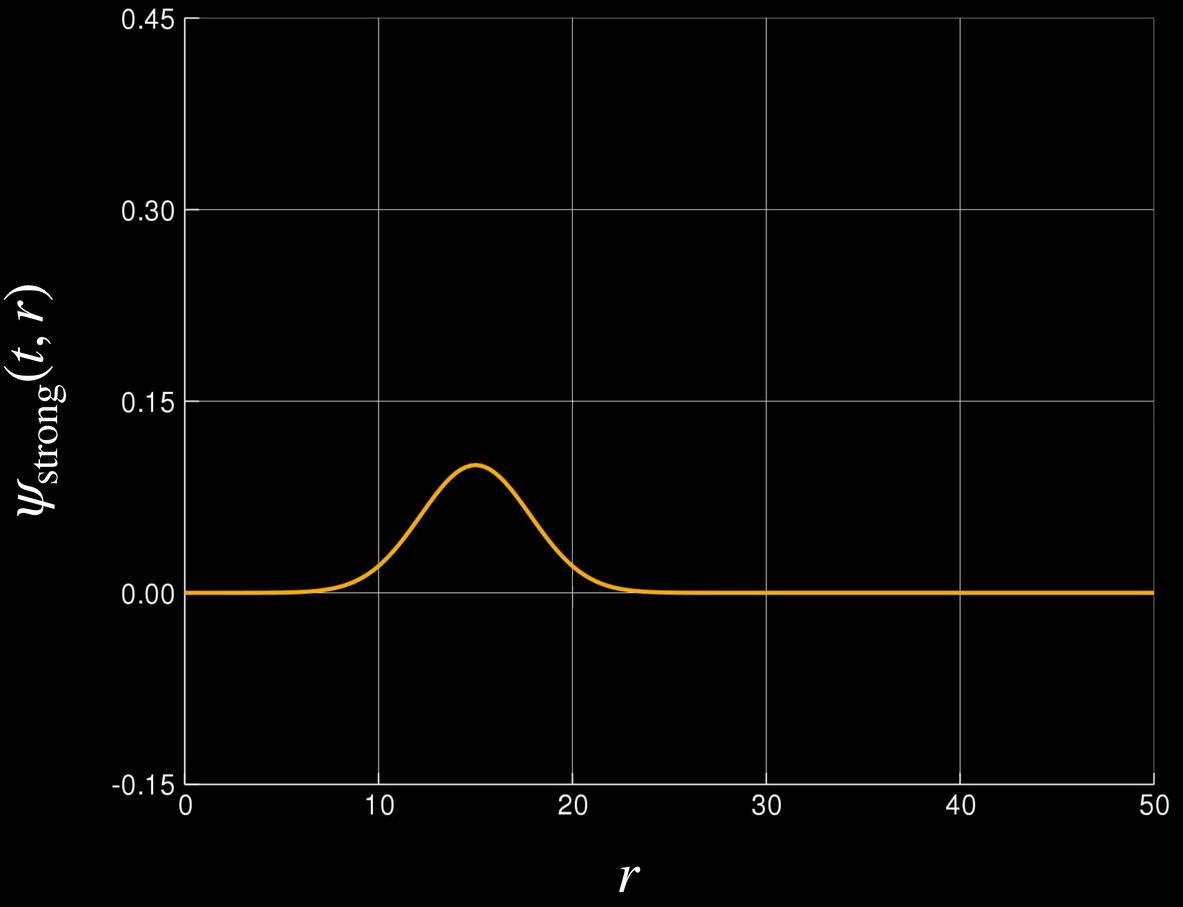
**In collaboration with Elcio Abdalla, Zach Etienne,
Bertha Cuadros-Melgar, and Carlos E. P. Oliveira**

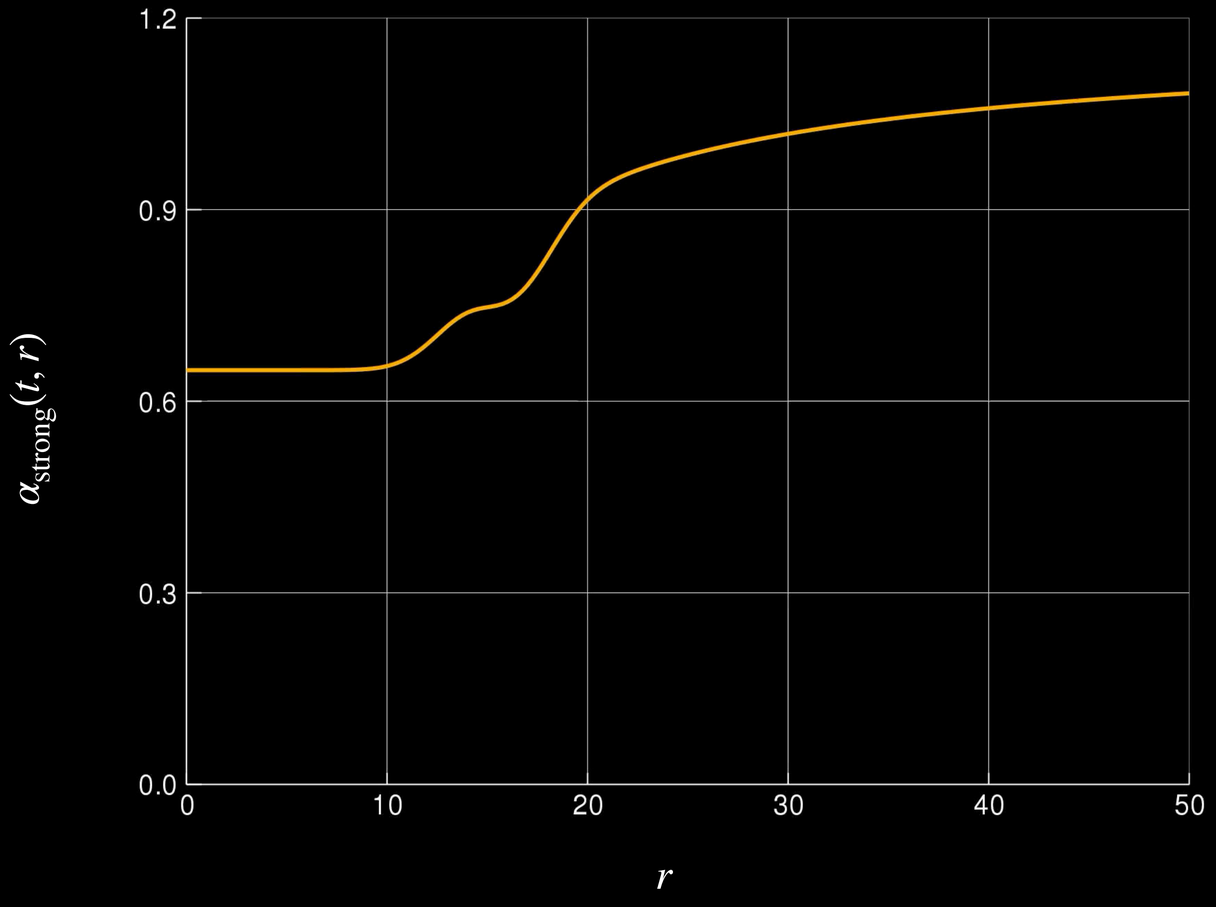
Motivation & Overview



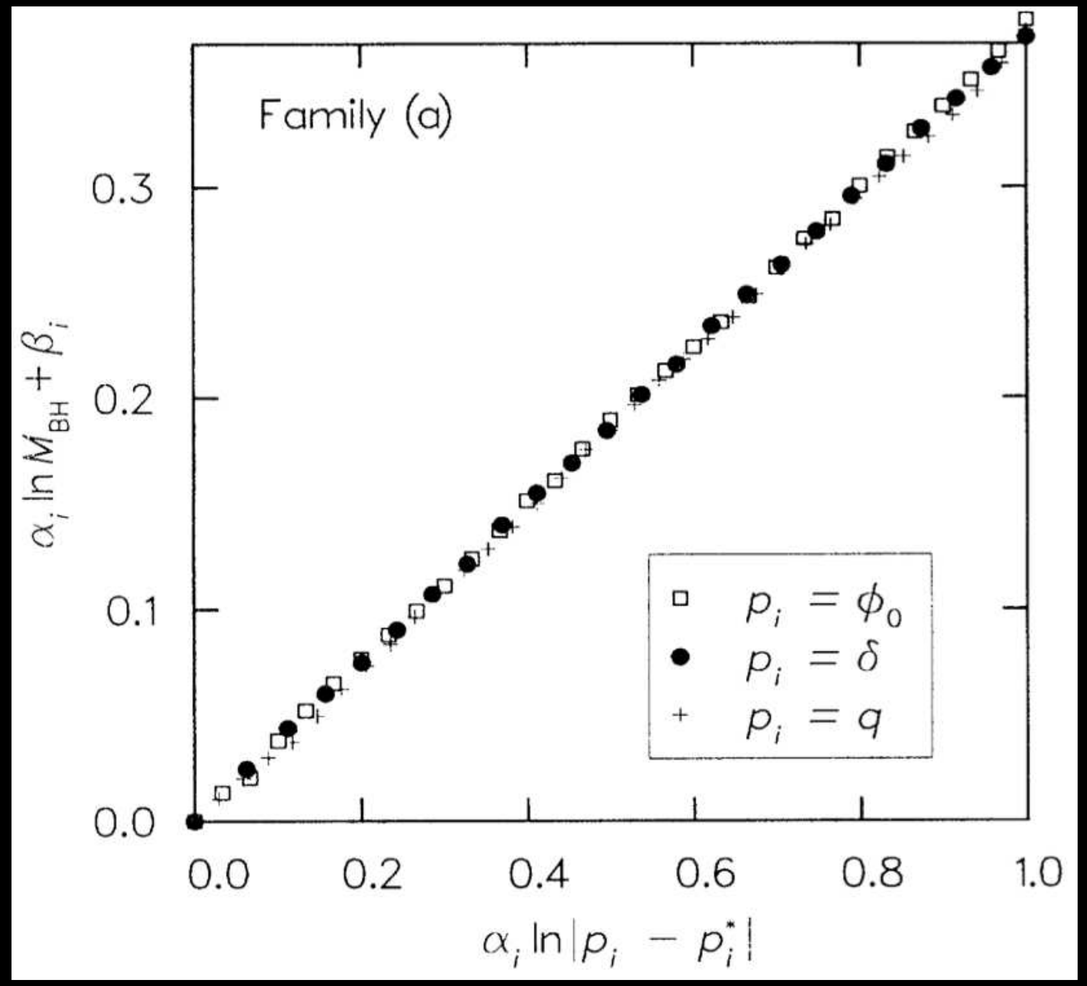


Motivation & Overview

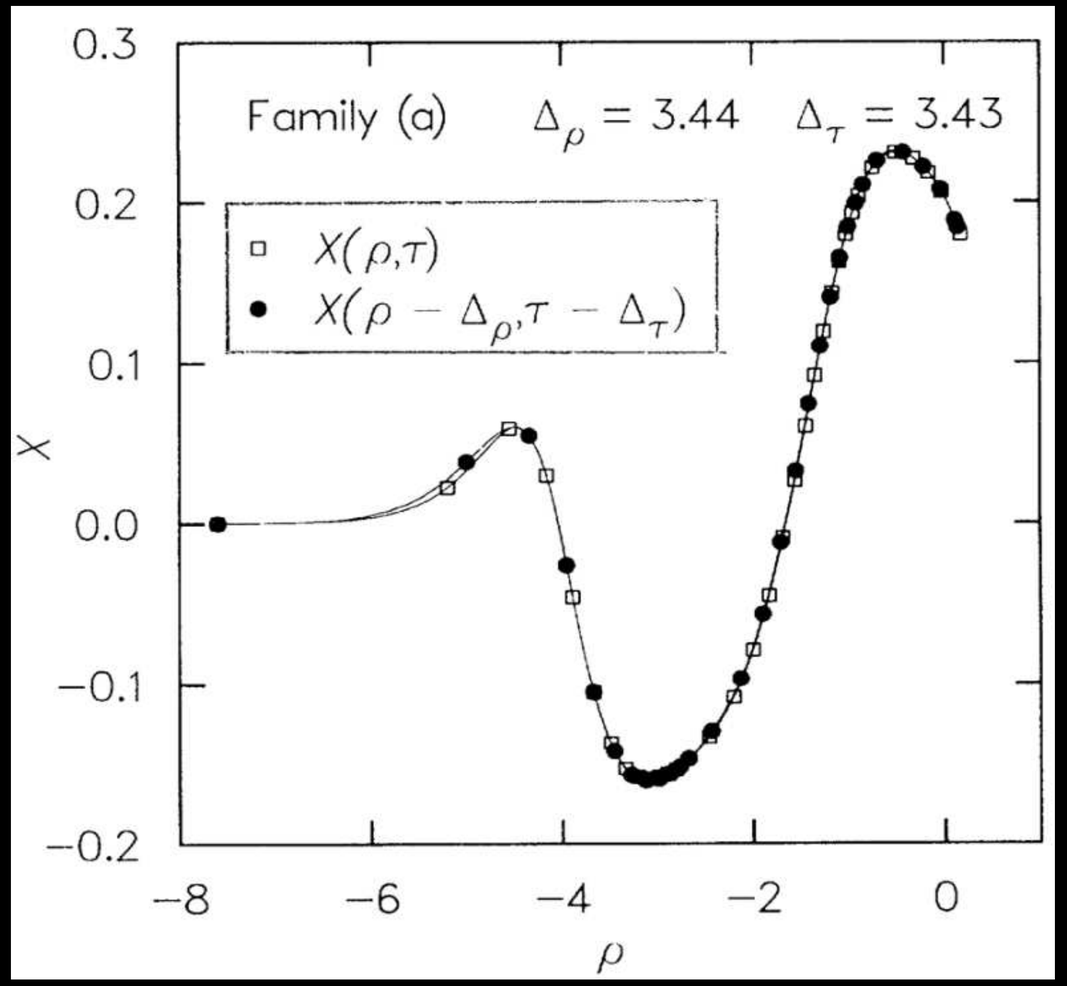




Motivation & Overview



Matt Choptuik. PRL 70 (1 Jan. 1993), pp. 9-12



Matt Choptuik. PRL 70 (1 Jan. 1993), pp. 9-12

$$M_{\mathrm{BH}} \propto \left| p - p^* \right|^{\gamma} \quad \gamma \approx 0.37$$

The SFcollapse1D code

- ★ Small (< 2000 lines of code), fast, open-source, well-documented, user-friendly code for gravitational collapse of massless scalar fields in Spherical-like coordinates

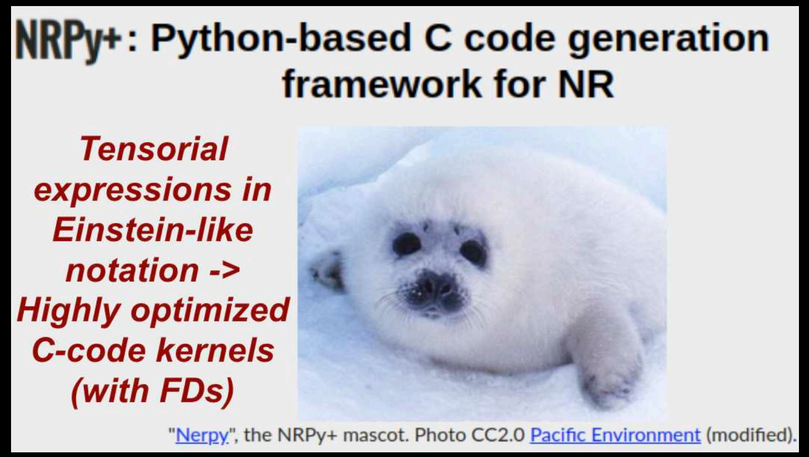
- ★ No AMR required!
  - ✓ Fully explores the symmetries of the problem
  - ✓ Smart choice of the numerical grid
  - ✓ Critical phenomena can be easily studied on the scale of laptop computers!

- ★ Available features
  - ✓ Multiple initial conditions available, easy to extend
  - ✓ Easy to extend grid choices
  - ✓ OpenMP parallelism
  - ✓ Useful diagnostics

Available at: https://github.com/leowerneck/SFcollapse1D

NRPy+ — Overview

- ★ NRPy+: "Python-based code generation for numerical relativity... and beyond!”



Available at: http://nrpyplus.net/

- ★ Similar to Kranc, but no Mathematica/Maple license required
- ★ Completely open-sourced, permissively licensed

Step 2: The right-hand side of $\partial_t \psi$ [Back to top]

Let us label each of the terms in the RHS of the $\partial_t \psi$ equation so that it is easier to understand their implementation:

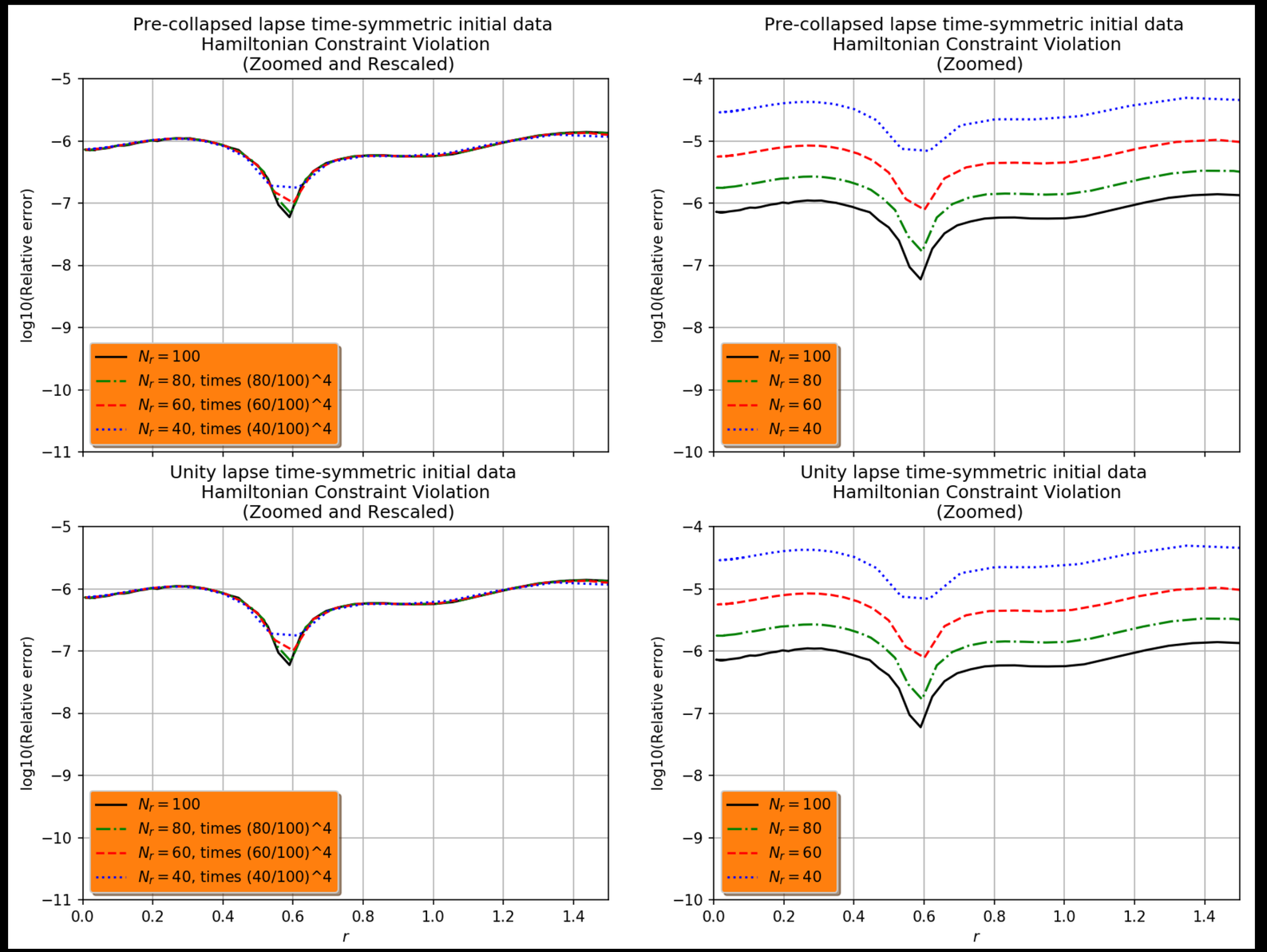
$$\partial_t \psi = \underbrace{-\alpha \Pi}_{\text{Term 1}} + \underbrace{\beta^i \partial_i \psi}_{\text{Term 2}} .$$

The first term is an advection term and therefore we need to set up the appropriate derivate. We begin by declaring the grid functions that we need to implement this equation.

A note on notation: We choose to declare the **s**calar **f**ield variable, ψ, as sf and the **s**calar **f**ield conjugate **M**omentum variable, Π, as sfM to avoid possible conflicts with other variables which might be commonly denoted by ψ or Π.

[2]:
```
# Step 2.a: Declare grid functions for psi and Pi
sf, sfM = gri.register_gridfunctions("EVOL",["sf", "sfM"])

# Step 2.a: Add Term 1 to sf_rhs: -alpha*Pi
sf_rhs = - alpha * sfM

# Step 2.b: Add Term 2 to sf_rhs: beta^{i}\partial_{i}\varphi
sf_dupD = ixp.declarerank1("sf_dupD")
for i in range(DIM):
    sf_rhs += betaU[i] * sf_dupD[i]
```

NRPy+ — Scalar field collapse



Overview of the codes

- Both codes have been designed with memory efficiency and user-friendliness in mind. The user is able to study critical phenomena using their desktop or laptop computer!
- **SFcollapse1D** is a great code for beginners
- **SFcollapse1D** can also be used for other spherically symmetric systems of interest
- **NRPy+CoI** is a flexible and powerful code. It allows the user to choose between many different curvilinear coordinate systems and thus can adapt to a plethora of different problems
- **NRPy+CoI** uses state-of-the-art numerical relativity, with NRPy+ being used to generate extremely efficient C code

Results

Scalar fields in curved spacetime — Initial conditions

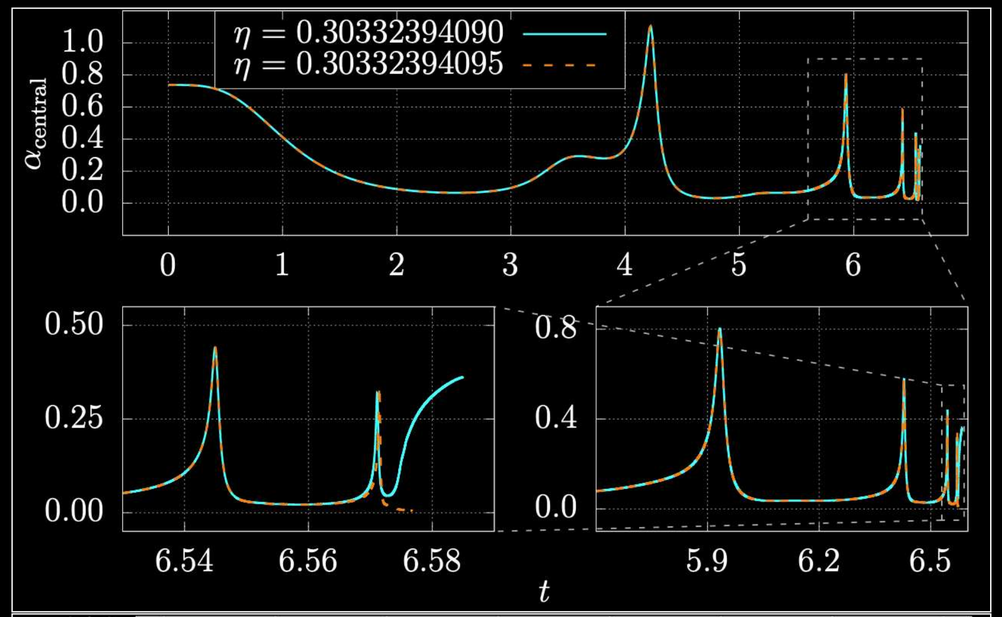
I $$\psi(0,r) = \eta \exp\left(-\frac{r^2}{\delta^2}\right)$$

II $$\psi(0,r) = \eta r^3 \exp\left[-\left(\frac{r-r_0}{\delta}\right)^2\right]$$

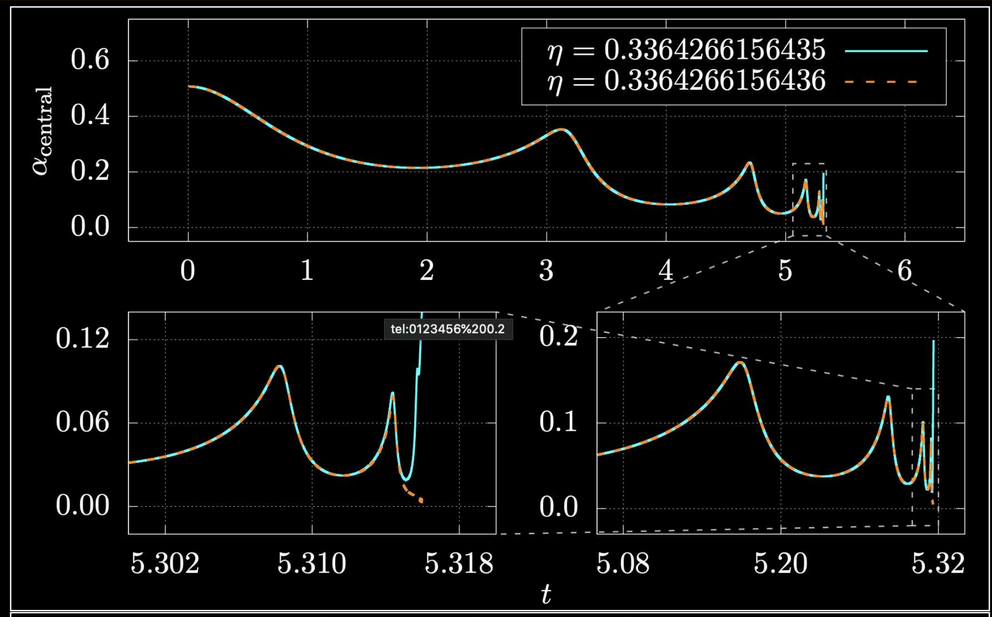
III $$\psi(0,r) = \eta\left\{1 - \tanh\left[-\left(\frac{r-r_0}{\delta}\right)^2\right]\right\}$$

Critical phenomena — Near critical lapse results

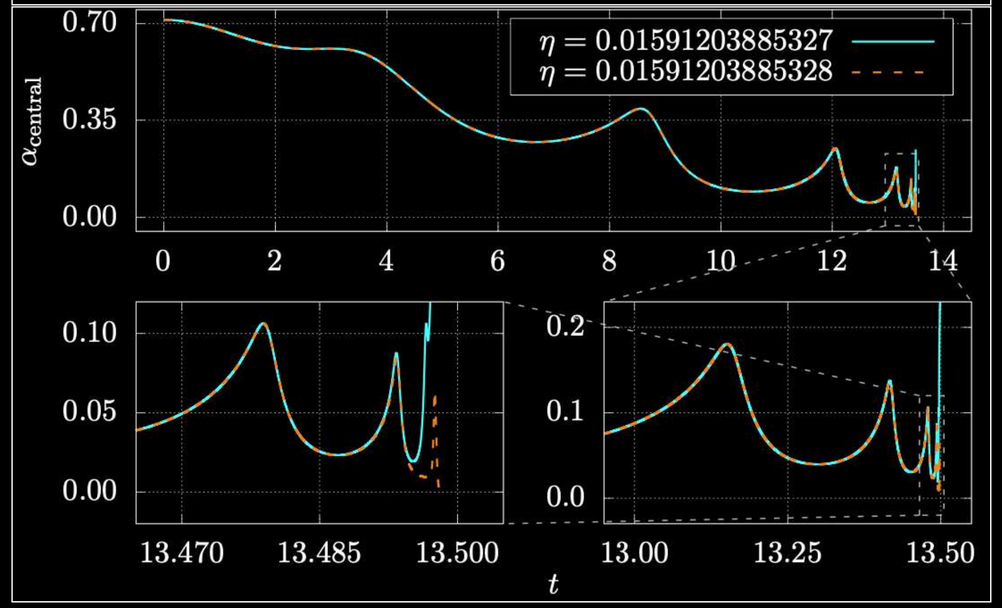
I BSSN



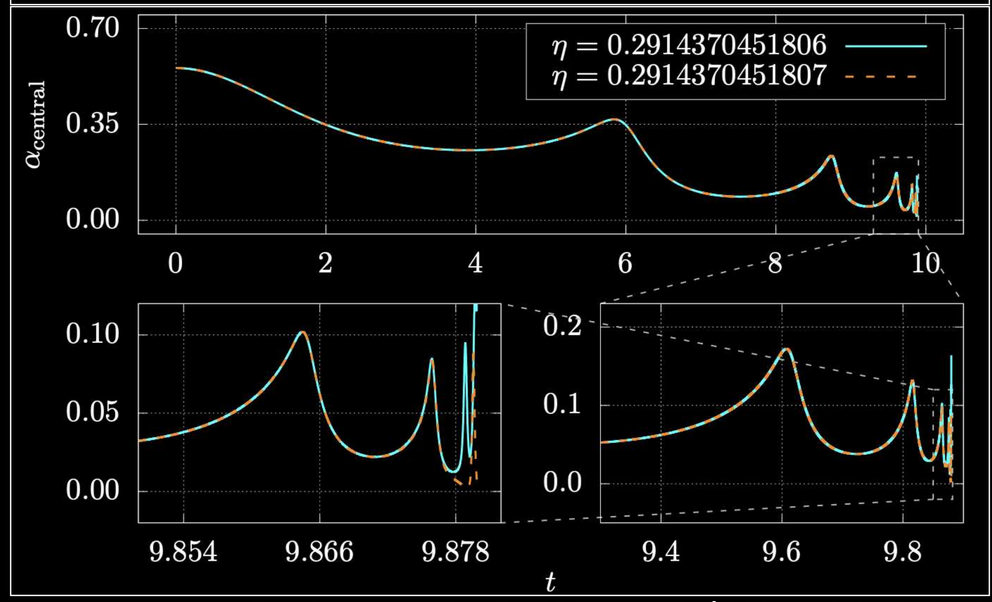
I ADM



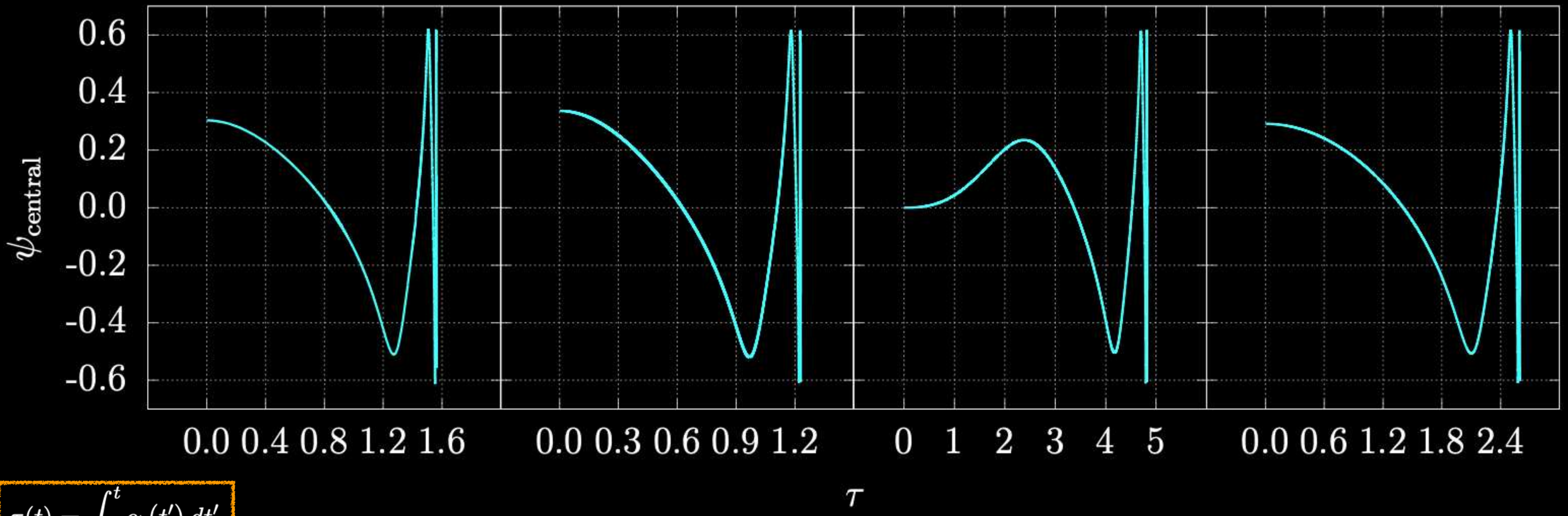
II ADM



III ADM



Critical phenomena — Universality



$$\tau(t) = \int_0^t \alpha\left(t'\right) dt'$$

Proper time

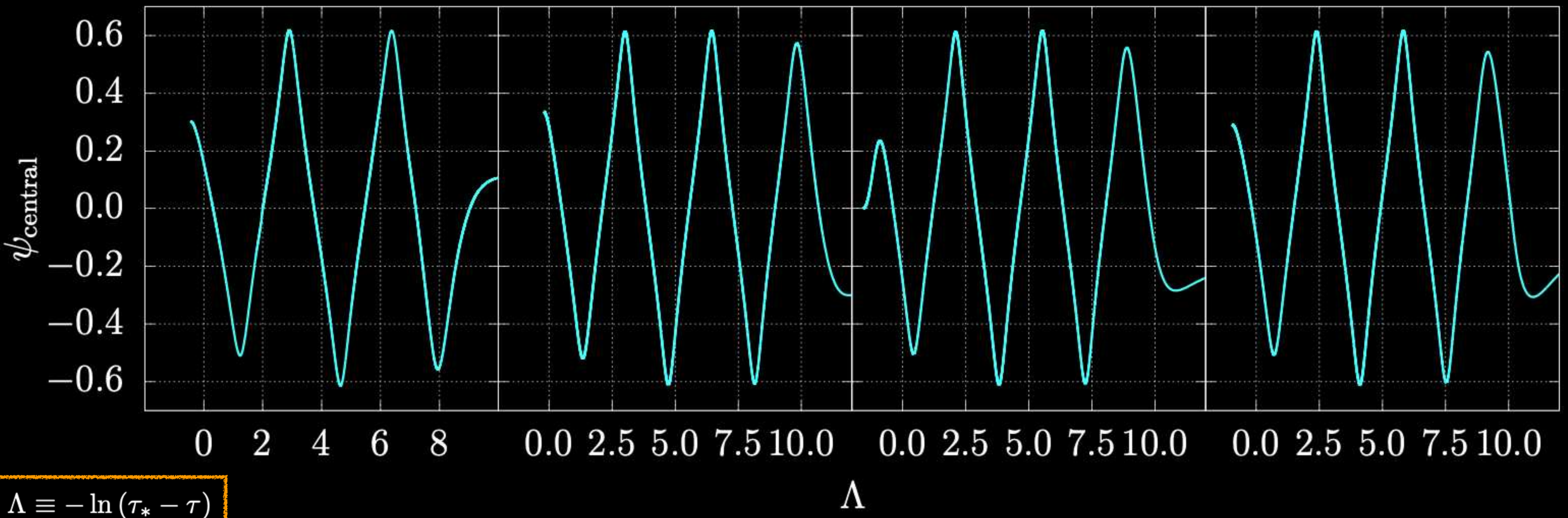
I
BSSN

I
ADM

II
ADM

III
ADM

Critical phenomena — Universality



$\Lambda \equiv -\ln(\tau_* - \tau)$

I BSSN

I ADM

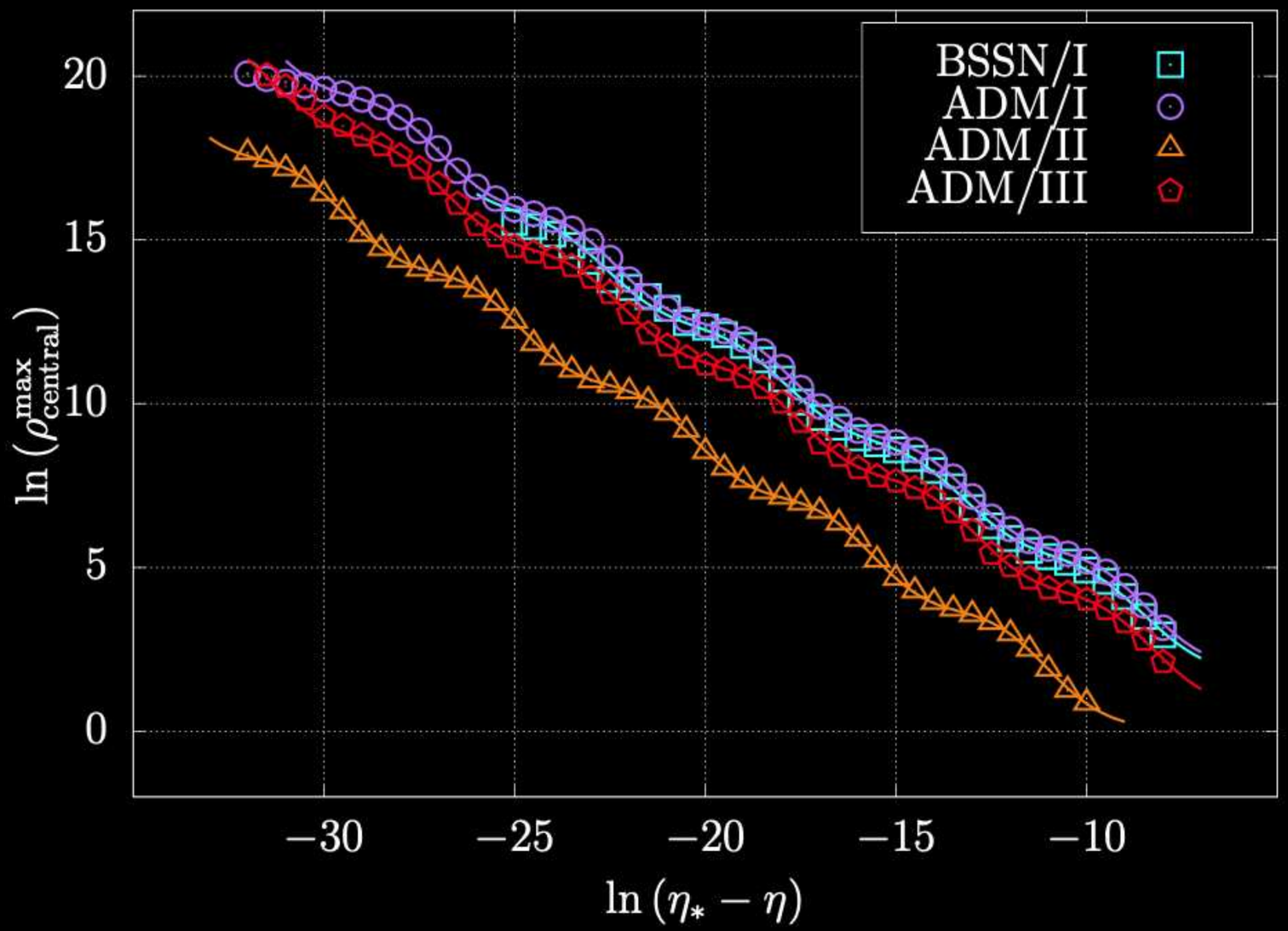
II ADM

III ADM

$\Delta = 3.46(2)$ $\Delta = 3.43(4)$ $\Delta = 3.42(4)$ $\Delta = 3.47(6)$

Critical phenomena — Universality

$$\ln\left(\rho^{\max}_{\text{central}}\right) = C - 2\gamma\, z + A \sin\left(\omega z + \varphi\right)$$



$$\omega = \frac{4\pi\gamma}{\Delta}$$

BSSN/I

$$\gamma = 0.374(3) \mid \Delta = 3.51^{+0.01}_{-0.06}$$

ADM′/I

$$\gamma = 0.375(1) \mid \Delta = 3.47^{+0.02}_{-0.04}$$

ADM/II

$$\gamma = 0.376(1) \mid \Delta = 3.45^{+0.02}_{-0.03}$$

ADM/III

$$\gamma = 0.375(1) \mid \Delta = 3.47^{+0.03}_{-0.05}$$

Summary and future work

- We have developed new, efficient codes to study critical phenomena
- Currently working on publish these results
- In the future: study problems where the spherical symmetry condition is relaxed
- Introduce a cosmological constant (undergrad student project is in progress)

Questions?

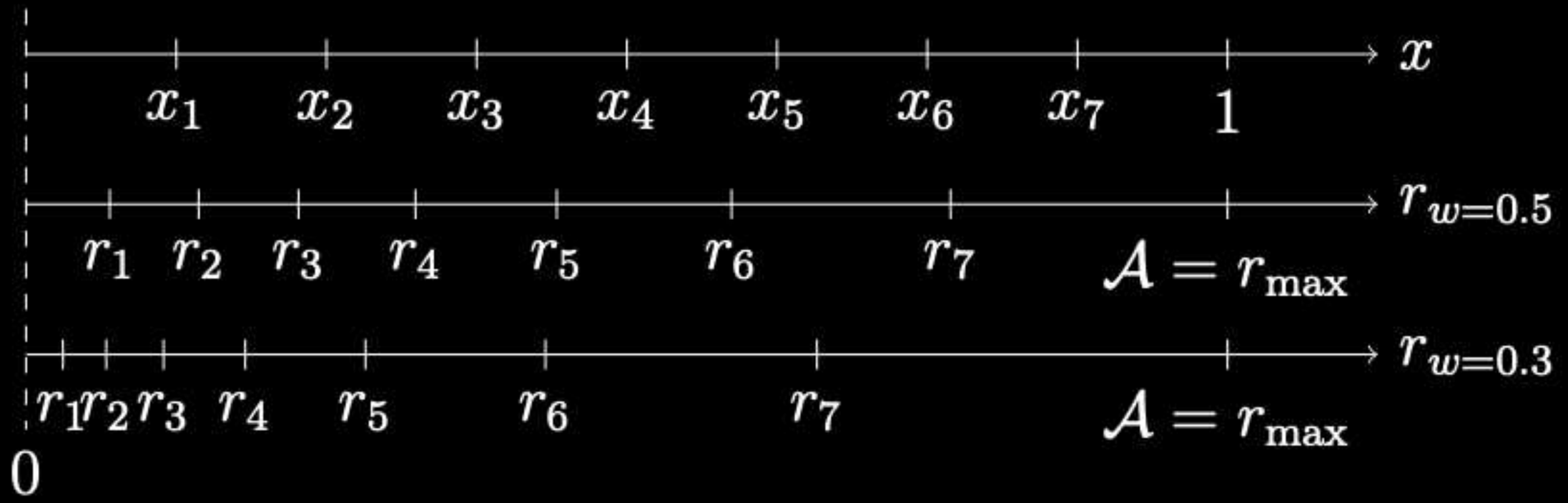
Motivation & Overview

- Main goal:
  - !! Study gravitational collapse of massless scalar fields
  - ?? Problem: our group had no experience with numerical relativity

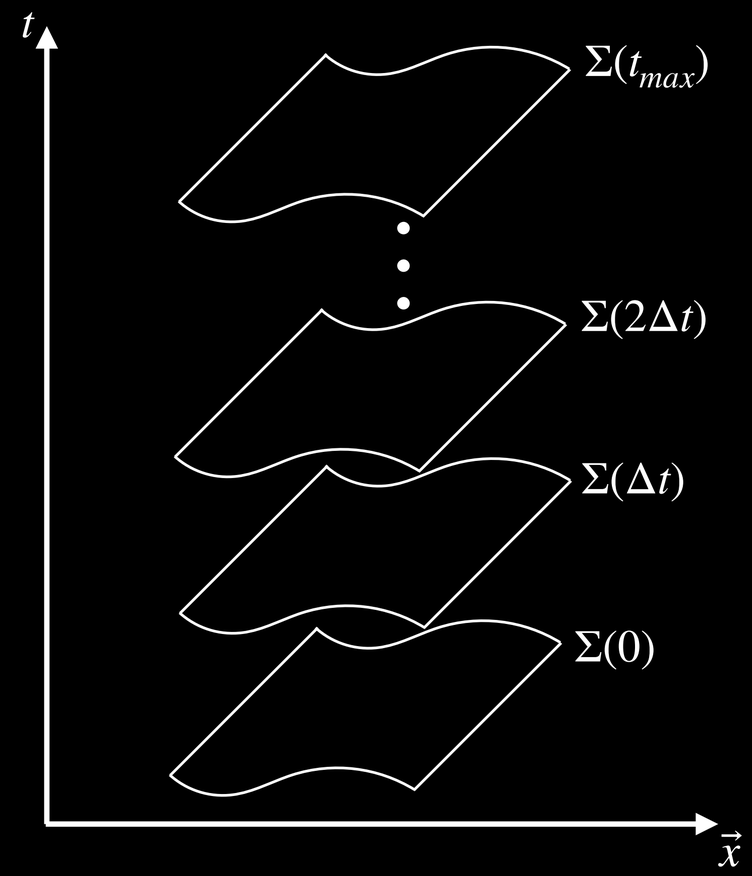
- Sub-goal (this talk!):
  - !? Develop a user-friendly, well-documented infrastructure to study gravitational collapse

Critical phenomena — SinhSpherical coordinates

$$r = A\frac{\sinh(x/w)}{\sinh(1/w)}$$



Numerical relativity



Scalar fields in curved spacetime — ADM formalism

$$ds^2 = -\alpha^2(t,r)dt^2 + a^2(t,r)dr^2 + r^2 d\Omega^2$$

$$\Psi(t,r) \equiv \partial_r \psi(t,r) \ , \ \Pi(t,r) \equiv \frac{a(t,r)}{\alpha(t,r)} \partial_t \psi(t,r)$$

$$\partial_t \Psi = \partial_r \left(\frac{\alpha}{a} \Pi \right)$$

$$\partial_t \Pi = \frac{1}{r^2} \partial_r \left(r^2 \frac{\alpha}{a} \Psi \right)$$

$$\frac{\partial_r a}{a} + \frac{a^2 - 1}{2r} = 2\pi r \left(\Psi^2 + \Pi^2 \right)$$

$$\frac{\partial_r \alpha}{\alpha} - \frac{\partial_r a}{a} - \frac{a^2 - 1}{r} = 0$$

Scalar fields in curved spacetime — BSSN formalism

$$\partial_t \psi = \beta^i \partial_i \psi - \alpha \Pi$$

$$\partial_t \Pi = \beta^i \partial_i \Pi + \alpha K \Pi - e^{-4\phi} \bar{\gamma}^{ij} \left(\partial_j \psi \partial_i \alpha - \alpha \bar{\Gamma}^k_{ij} \partial_k \psi + \alpha \partial_i \partial_j \psi + 2\alpha \partial_j \psi \partial_i \phi \right)$$

$$W \equiv e^{\phi}$$

$$\hat{D}^2 W = -2\pi W^5 \rho$$

The SFcollapse1D code

$$\frac{A_{j+1}^{n+1} - A_j^{n+1}}{\Delta r} + \frac{\exp\left(A_{j+1}^{n+1} + A_j^{n+1}\right) - 1}{2r_{j+1/2}} - 2\pi r_{j+1/2}\left\{\left[\frac{1}{2}\left(\Phi_{j+1}^{n+1} + \Phi_j^{n+1}\right)\right]^2 + \left[\frac{1}{2}\left(\Pi_{j+1}^{n+1} + \Pi_j^{n+1}\right)\right]^2\right\} = 0 \ ,$$

```
/* Function to solve the Hamiltonian constraint */
real evolution::pointwise_solution_of_the_Hamiltonian_constraint( const int j, grid::parameters grid, const realvec Phi,

  DECLARE_GRID_PARAMETERS;

  /* Set auxiliary variables */
  const real A         = log(a[j-1]);              // A^{n+1}_{j}
  const real avgPhi    = 0.5*( Phi[j] + Phi[j-1] ); // 0.5*( Phi^{n+1}_{j+1} + Phi^{n+1}_{j} )
  const real avgPi     = 0.5*( Pi[j]  + Pi[j-1]  ); // 0.5*(  Pi^{n+1}_{j+1} +  Pi^{n+1}_{j} )
  const real PhiSqr    = SQR(avgPhi);
  const real PiSqr     = SQR(avgPi);
  const real midx0     = 0.5 * ( x[0][j] + x[0][j-1] );
#if( COORD_SYSTEM == SPHERICAL )
  const real PhiPiTerm = 2.0 * M_PI * midx0 * ( PhiSqr + PiSqr );
  const real half_invr = 0.5 / midx0;
#elif( COORD_SYSTEM == SINH_SPHERICAL )
  const real PhiPiTerm = 2.0 * M_PI * (SQR(sinhA) * inv_sinhW) * sinh(midx0*inv_sinhW) * cosh(midx0*inv_sinhW) / SQR(sinh_inv_W) * ( PhiSqr + PiSqr );
  const real half_invr = 0.5/( sinhW * tanh(midx0*inv_sinhW) );
#else
  utilities::SFcollapse1D_error(COORD_SYSTEM_ERROR);
#endif

  /* Set Newton's guess */
  real A_old = log(a[j-1]);

  /* Set variable for output */
  real A_new = A_old;

  /* Set a counter */
  real iter = 0;

  /* Perform Newton's method */
  do{

    /* Update A_old */
    A_old = A_new;

    /* Compute f and df */
    const real tmp0 = half_invr * exp(A_old+A);
    const real f  = inv_dx0 * (A_old - A) + tmp0 - half_invr - PhiPiTerm;
    const real df = inv_dx0 + tmp0;

    /* Update A_new */
    A_new = A_old - f/df;

    /* Increment the iterator */
    iter++;

  } while( ( fabs(A_new - A_old) > NEWTON_TOL ) && ( iter <= NEWTON_MAX_ITER ) );

  /* Check for convergence */
  if( iter > NEWTON_MAX_ITER ) cerr << "\n(Newton's method WARNING) Newton's method did not converge to a root! j = " << j << " | iter = " << iter << endl;

  /* Return the value of a */
  return( exp(A_new) );
}
```

Adopts spherical coordinates, with both uniform and non-uniform radial sampling!

NRPy+ — Overview

- Input: Einstein notation + simple Python data structures (lists)
- Output: Highly efficient C code with CSE, SIMD, and finite differences

Common Subexpression Elimination (CSE)

```
[1]: from outputC import outputC
     import sympy as sp

     # Declare some variables, using SymPy's symbols() function
     a,b,c = sp.symbols("a b c")

     # Set x = b*sin(2*a) + c/sin(2*a)
     x = b*sp.sin(2*a) + c/( sp.sin(2*a) )

     outputC(x,"x")
```

```
/*
 *  Original SymPy expression:
 *  "x = b*sin(2*a) + c/sin(2*a)"
 */
{
   const double tmp_0 = sin(2*a);
   x = b*tmp_0 + c/tmp_0;
}
```

Single-Instruction, Multiple Data (SIMD) compiler intrinsics

```
[2]: # Initialize code Python/NRPy+ modules
     import sympy as sp
     import NRPy_param_funcs as par
     from outputC import outputC

     # Declare some variables, using SymPy's symbols function
     a,b,c = sp.symbols("a b c")

     # Set x = b*sin(2*a) + c/sin(2*a)
     x = b*sp.sin(2*a) + c/( sp.sin(2*a) )

     outputC(x,"x",params="SIMD_enable=True")
```

```
/*
 *  Original SymPy expression:
 *  "x = b*sin(2*a) + c/sin(2*a)"
 */
{
   const double tmp_Integer_1 = 1.0;
   const REAL_SIMD_ARRAY _Integer_1 = ConstSIMD(tmp_Integer_1);

   const double tmp_Integer_2 = 2.0;
   const REAL_SIMD_ARRAY _Integer_2 = ConstSIMD(tmp_Integer_2);

   const double tmp_NegativeOne_ = -1.0;
   const REAL_SIMD_ARRAY _NegativeOne_ = ConstSIMD(tmp_NegativeOne_);

   const REAL_SIMD_ARRAY tmp_0 = SinSIMD(MulSIMD(_Integer_2, a));
   x = FusedMulAddSIMD(b, tmp_0, DivSIMD(c, tmp_0));
}
```

NRPy+ — Scalar field collapse

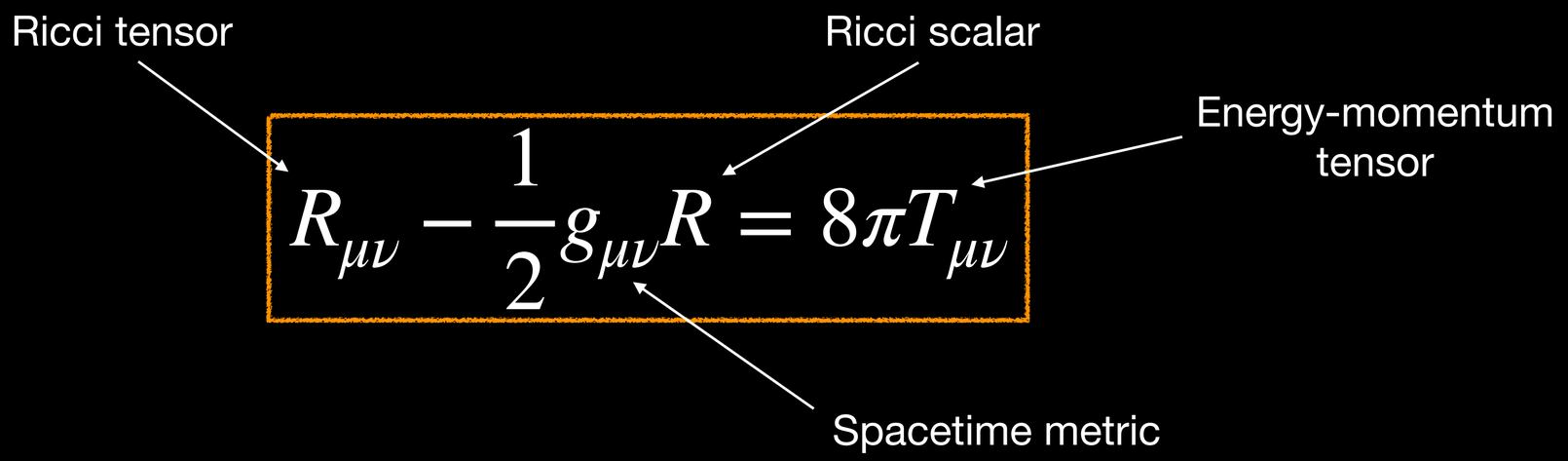
[3]:
```
import sympy as sp
from outputC import lhrh
import grid as gri
import indexedexp as ixp
import NRPy_param_funcs as par
import finite_difference as fin

# Set the spatial dimension to 1
par.set_parval_from_str("grid::DIM",1)

# Set the finite difference accuracy to second order
par.set_parval_from_str("finite_difference::FD_CENTDERIVS_ORDER",2)

# Register the input gridfunction "phi" and the gridfunction to which date is output, "output"
phi, output = gri.register_gridfunctions("AUX",["phi","output"])

# Declare phi_dDD: a rank-2 indexed expression: phi_dDD[i][j] := \partial_{i}\partial_{j}\phi
phi_dDD = ixp.declarerank2("phi_dDD","sym01")

# Set output to \partial_{0}^{2}\phi
output = phi_dDD[0][0]

# Output to the screen the core C code for evaluating the finite difference derivative
fin.FD_outputC("stdout",lhrh(lhs=gri.gfaccess("out_gf","output"),rhs=output))
```

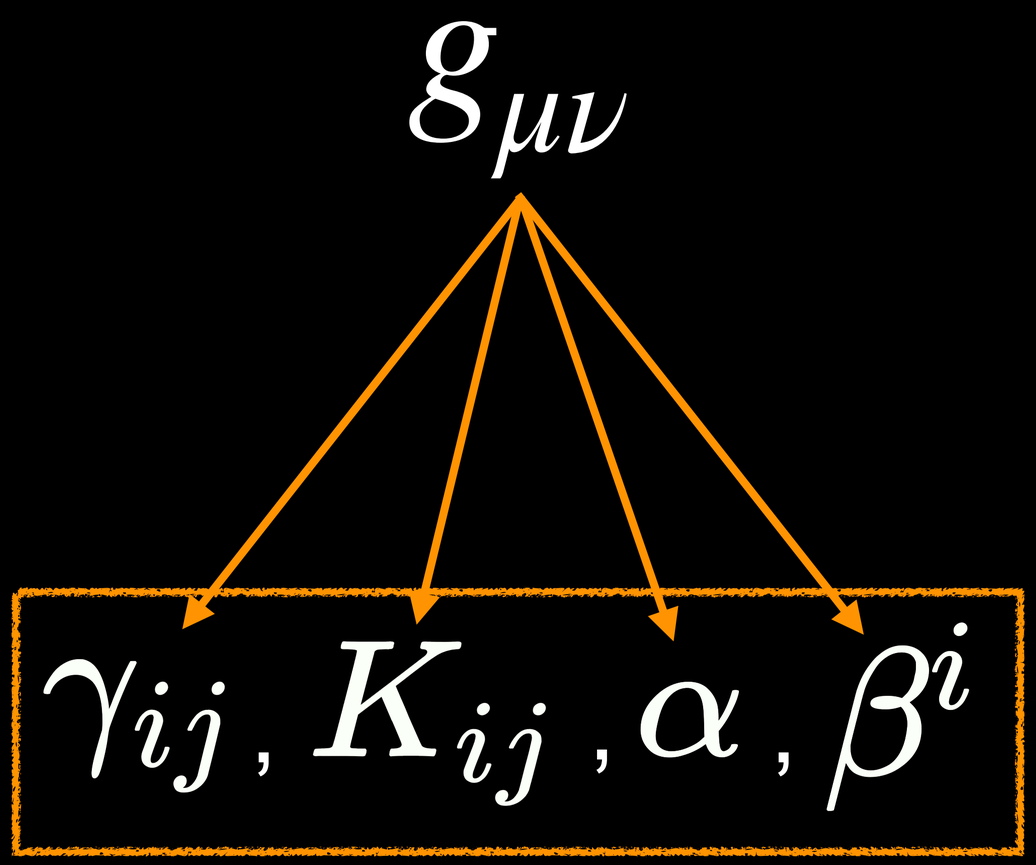
```
{
  /*
   * NRPy+ Finite Difference Code Generation, Step 1 of 2: Read from main memory and compute finite difference stencils:
   */
  /*
   *  Original SymPy expression:
   *  "const double phi_dDD00 = invdx0**2*(-2*phi + phi_i0m1 + phi_i0p1)"
   */
  const double phi_i0m1 = aux_gfs[IDX2(PHIGF, i0-1)];
  const double phi = aux_gfs[IDX2(PHIGF, i0)];
  const double phi_i0p1 = aux_gfs[IDX2(PHIGF, i0+1)];
  const double FDPart1_Integer_2 = 2.0;
  const double phi_dDD00 = ((invdx0)*(invdx0))*(-FDPart1_Integer_2*phi + phi_i0m1 + phi_i0p1);
  /*
   * NRPy+ Finite Difference Code Generation, Step 2 of 2: Evaluate SymPy expressions and write to main memory:
   */
  /*
   *  Original SymPy expression:
   *  "aux_gfs[IDX2(OUTPUTGF, i0)] = phi_dDD00"
   */
  aux_gfs[IDX2(OUTPUTGF, i0)] = phi_dDD00;
}
```

Computing $\partial_x^2 \phi$ using NRPy+

Dr. Leonardo R. Werneck | West Virginia University

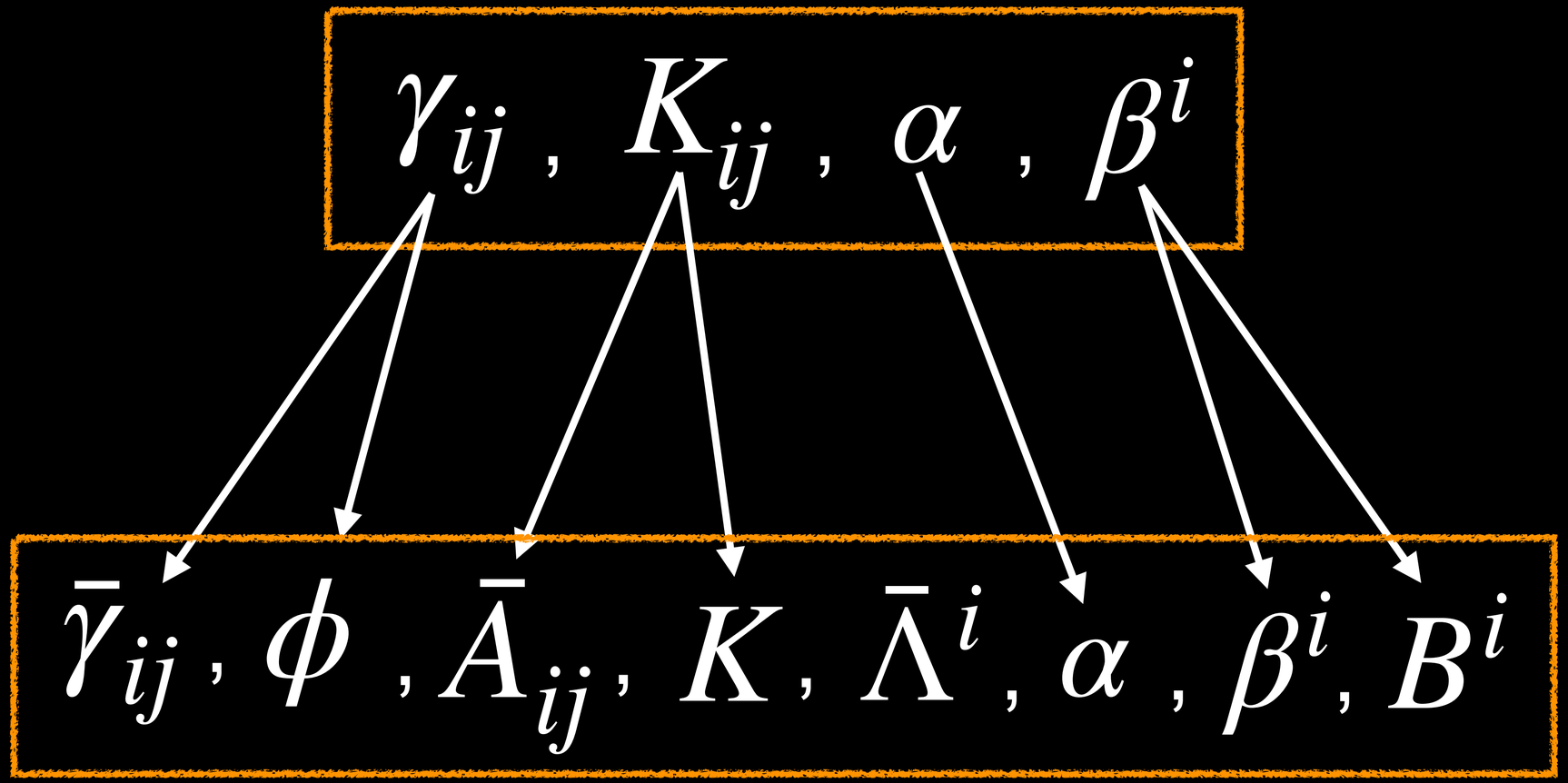
Numerical relativity





The ADM variables

The ADM variables



The BSSN variables

The ADM formalism

$$P^{\mu}_{\ \nu} = \gamma^{\mu}_{\ \nu} \equiv \delta^{\mu}_{\ \nu} + n^{\mu} n_{\nu} \qquad t^{\mu} \equiv \alpha n^{\mu} + \beta^{\mu}$$

$$n_{\mu} = (-\alpha, 0, 0, 0)$$

$$n^{\mu} = (\alpha^{-1}, -\alpha^{-1}\beta^{i})$$

$$ds^2 = -\alpha^2 dt^2 + \gamma_{ij}\left(dx^i + \beta^i dt\right)\left(dx^j + \beta^j dt\right)$$

The (3+1) ADM metric (line element)

$$\mathscr{H} \equiv {}^{(3)}R + K^2 - K_{ij}K^{ij} - 16\pi\rho = 0$$

The Hamiltonian constraint

$$\mathscr{M}_i \equiv D_j K^j_{\ i} - D_i K - 8\pi S_i = 0$$

The momentum constraint

$$\partial_t \gamma_{ij} = -2\alpha K_{ij} + D_i\beta_j + D_j\beta_i$$

Spatial metric evolution equation

$$\partial_t K_{ij} = \beta^{\ell}\partial_{\ell} K_{ij} + K_{i\ell}\partial_j\beta^{\ell} + K_{j\ell}\partial_i\beta^{\ell} - D_i D_j\alpha$$
$$+\alpha\left({}^{(3)}R_{ij} + KK_{ij} - 2K_{i\ell}K^{\ell}_{\ j}\right) + 4\pi\alpha\left[\gamma_{ij}\left(S-\rho\right) - 2S_{ij}\right]$$

Extrinsic curvature evolution equation

The BSSN formalism

$$\bar{\gamma}_{ij} = e^{-4\phi}\gamma_{ij}$$

$$\bar{\gamma}_{ij} = \hat{\gamma}_{ij} + \epsilon_{ij}$$

$$\partial_t \bar{\gamma} = 0$$

$$K_{ij} = A_{ij} + \frac{1}{3}\gamma_{ij}K$$

$$\bar{A}_{ij} = e^{-4\phi}A_{ij}$$

$$\bar{A}^i{}_i = \bar{\gamma}^{ij}\bar{A}_{ij} = 0$$

$$\Delta^i_{jk} \equiv \bar{\Gamma}^i_{jk} - \hat{\Gamma}^i_{jk}$$

$$\Delta^i \equiv \bar{\gamma}^{jk}\Delta^i_{jk}$$

$$\mathscr{C}^i \equiv \bar{\Lambda}^i - \Delta^i = 0$$

The BSSN formalism

$$\partial_\perp \bar{\gamma}_{ij} = -\frac{2}{3}\bar{\gamma}_{ij}\bar{D}_k\beta^k - 2\alpha\bar{A}_{ij}$$

$$\partial_\perp \equiv \partial_t - \mathcal{L}_\beta$$

$$\partial_\perp \bar{A}_{ij} = -\frac{2}{3}\bar{A}_{ij}\bar{D}_k\beta^k - 2\alpha\bar{A}_{ik}\bar{A}^k{}_j + \alpha\bar{A}_{ij}K$$
$$+ e^{-4\phi}\Big[-2\alpha\bar{D}_i\bar{D}_j\phi + 4\alpha\bar{D}_i\phi\bar{D}_j\phi + 4\bar{D}_{(i}\alpha\bar{D}_{j)}\phi$$
$$- \bar{D}_i\bar{D}_j\alpha + \alpha(\bar{R}_{ij} - 8\pi S_{ij})\Big]^{\mathrm{TF}}$$

$$\partial_\perp \phi = \frac{1}{6}\bar{D}_k\beta^i - \frac{1}{6}\alpha K$$

$$\partial_\perp K = \frac{\alpha}{3}K^2 + \alpha\bar{A}_{ij}\bar{A}^{ij}e^{-4\phi}(\bar{D}^2\alpha + 2\bar{D}^i\alpha\bar{D}_i\phi) + 4\pi\alpha(\rho + S)$$

$$\partial_\perp \bar{\Lambda}^i = \bar{\gamma}^{jk}\hat{D}_j\hat{D}_k\beta^i + \frac{2}{3}\Delta^i\bar{D}_j\beta^j + \frac{1}{3}\bar{D}^i\bar{D}_j\beta^j - \frac{4}{3}\alpha\bar{\gamma}^{ij}\partial_j K$$
$$- 2\bar{A}^{jk}(\delta^i{}_j\partial_k\alpha - 6\alpha\delta^i{}_j\partial_k\phi - \alpha\Delta^i_{jk}) - 16\pi\alpha\bar{\gamma}^{ij}S_j.$$

Brown, Phys.Rev.D79:104029,2009, arXiv: 0902.3652 Baumgarte *et al.*, Phys.Rev.D.87.044026,2012, arXiv: 1211.6632

The BSSN formalism

$$\partial_t \alpha = \beta^j \bar{D}_j \alpha - 2\alpha K \ ,$$

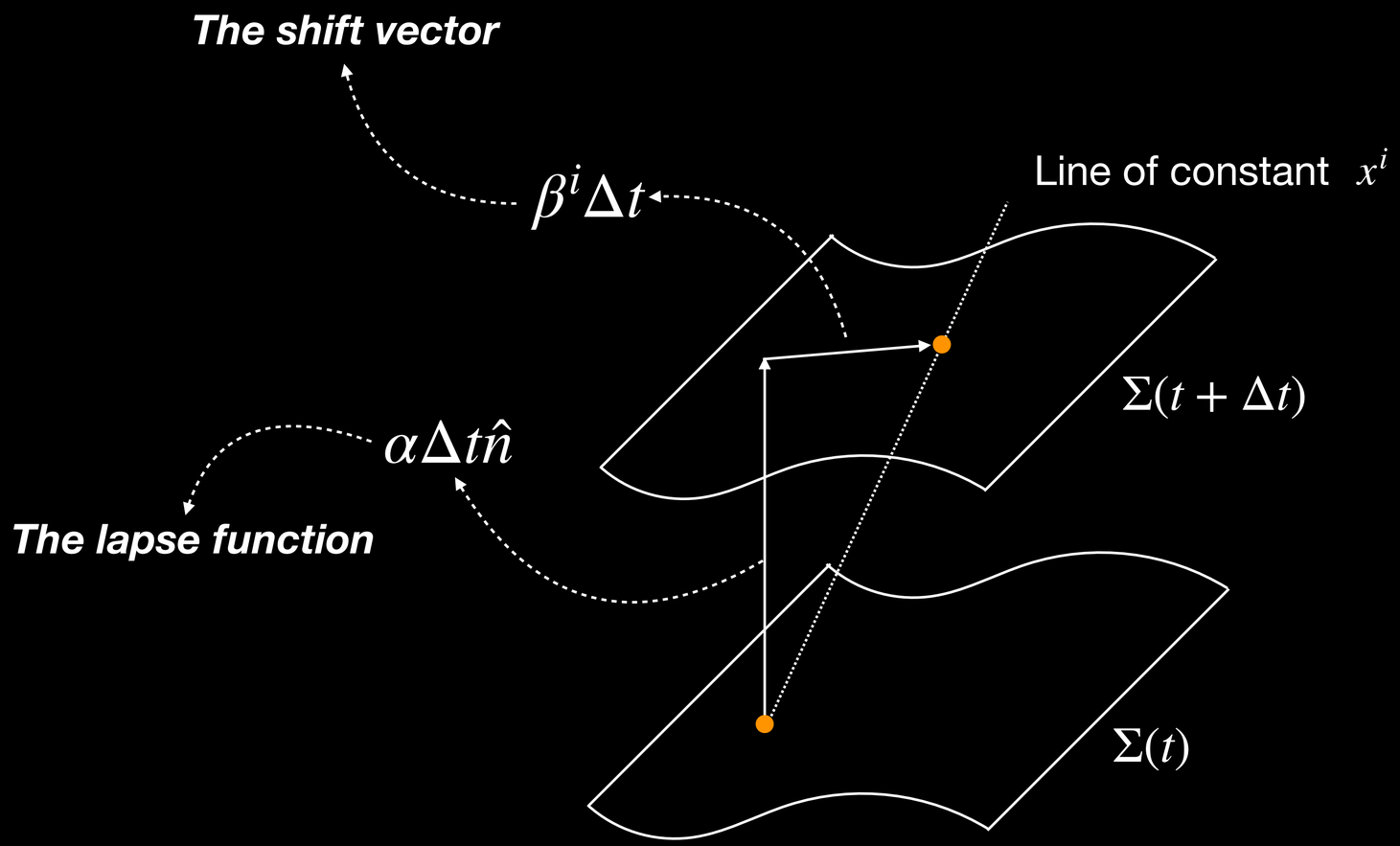
Bona *et al.*, PRL 75.4 (1995) gr-qc/9412071

$$\partial_t \beta^i = \beta^j \bar{D}_j \beta^i + \frac{3}{4} B^i \ ,$$

Alcubierre, PRD 67.8 (2003) p. 084023, gr-qc/0206072

$$\partial_t B^i = \beta^j \bar{D}_j B^i + \left(\partial_t \bar{\Lambda}^i - \beta^j \bar{D}_j \bar{\Lambda}^i \right) - \eta_S B^i$$

The lapse and the shift



The extrinsic curvature