Developing a solver for inelastic neutrino-nucleon scattering in core-collapse supernovae

Program in Applied and Computational Mathematics

Advised by Prof. Adam Burrows

Connor Hainje, 2021 Apr 20

Introduction

Neutrino scattering in core-collapse supernovae

- Massive stars explode in their death => known as “core-collapse supernovae”
- Explosions are very complex nuclear physics processes
- Dominant neutrino-matter interactions are absorption processes which are well understood
- Inelastic neutrino-electron and neutrino-nucleon scattering heats the matter at a lower rate, ~10% to 20% of the total heating
- Inelastic neutrino-nucleon scattering often not included due to lack of well-described, efficient description of its effects, but could have larger impact than neutrino-electron
- Wang & Burrows 2020 presents a novel Kompaneets formalism for describing this scattering

Kompaneets formalism

from Wang and Burrows 2020

- Describe the spectrum of neutrinos by the **distribution function** $f(\varepsilon)$, with ε the **neutrino energy** (in MeV)
- Define the **dimensionless neutrino energy** $x \equiv \varepsilon / kT$
- Introduce the **energy flux** $I_\nu = \alpha x^6 \left[\frac{df}{dx} + f(x) - f(x)^2 \right]$
- The time evolution of f is then described by the following ODE

$$\frac{df}{dt} = \frac{1}{x^2} \frac{dI_\nu}{dx}$$

- We want to build a numerical solver that implements this formalism!

Kompaneets formalism

from Wang and Burrows 2020

- This formalism naturally conserves neutrino number
  - Differential neutrino number $\propto x^2 \frac{df}{dt} = \frac{dI_\nu}{dx}$
  - Thus, change in neutrino number $= \int_0^\infty \frac{dI_\nu}{dx}\, dx = I_\nu(\infty) - I_\nu(0) = 0$
  - We have to preserve this in our solver!

Reformulation of ODE

- Most differentiation schemes prefer equispaced binning, but we are going to use logarithmically spaced bins
- Can transform to equispaced binning in $\log x$ by compute derivative as

$$\frac{1}{x^2}\frac{dI_\nu}{dx} = \frac{1}{x^3}\frac{dI_\nu}{d(\log x)}$$

- Thus, we can rewrite the ODE as

$$\frac{d(x^3 f)}{dt} = \frac{dI_\nu}{d(\log x)}$$

Reformulation of ODE

- We can also rewrite I_ν to depend only on $x^3 f$ (for brevity, call it y)

$$
\begin{aligned}
I_\nu &= \alpha x^6 \left[\frac{df}{dx} + f - f^2 \right] \\
&= \alpha \left[x^3 \left(\frac{dy}{dx} - 3x^2 f(x) \right) + x^3 y - y^2 \right] \\
&= \alpha \left[x^2 \frac{dy}{d(\log x)} + (x^3 - 3x^2) y - y^2 \right]
\end{aligned}
$$

Pipeline

- Take in sampling points x_i, sampled values f_i, physical parameters
- Compute $y_i = x_i^3 f_i$
- Compute I_ν
- Compute $\dfrac{dI_\nu}{d(\log x)}$
- Update y_i values via an ODE stepper
- Update $f_i = y_i / x_i^3$

Numerical neutrino conservation

- Recall that the change in neutrino number $\propto \int \frac{dI_\nu}{dx} dx$
- For discretized energy bins with *edges* x_i, this is

$$\sum I_\nu(x_{i+1}) - I_\nu(x_i) = I_\nu(x_{\text{max}}) - I_\nu(x_{\text{min}})$$

- Thus, we compute I_ν on the bin *edges* and manually set it equal to zero on outermost edges

Pipeline (revised)

- Take in sampling points x_i (bin *centers*), sampled values f_i, physical parameters
- Compute $y_i = x_i^3 f_i$ on bin *centers*
- Compute I_ν on the bin *edges*
  - Need $\dfrac{dy}{d(\log x)}$, $y(x)$ on bin *edges*
- Compute $\dfrac{dI_\nu}{d(\log x)}$ on bin *centers*
- Perform updates via an ODE stepper

- $d/d(\log x)$ calculations require differentiation methods
- $y(x)$ on bin edges requires interpolation methods
- Specifically want methods that will give us the values on the midpoints of the intervals

Linear methods

Method overview

- Given two points (x_i, y_i) and (x_{i+1}, y_{i+1}), we find the linear Lagrange interpolant and its derivative
- At the interval midpoint, these take values

$$y(x_{i+1/2}) = \frac{1}{2}\left(y_i + y_{i+1}\right)$$

$$y'(x_{i+1/2}) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Linear methods

Analytic error performance

- Can re-derive the method by Taylor expansions to find truncation error
- Because x is the interval midpoint

$$y_{\{i+1,i\}} = f(x \pm h) = f(x) \pm hf'(x) + \frac{1}{2}h^2 f''(x) + \mathcal{O}(h^3)$$

- Find

$$f(x) = \frac{f(x-h) + f(x+h)}{2} + h^2 f''(x) + \mathcal{O}(h^4)$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{1}{6}h^2 f'''(x) + \mathcal{O}(h^4)$$

Cubic Lagrange methods

Method overview

- We can instead take two points *as well as* the next two closest points, i.e. (x_j, y_j) for $j \in \{i-1,\ i,\ i+1,\ i+2\}$, and write the cubic Lagrange interpolant (and its derivative)

- Using the fact that the bins are equispaced and that we desire x at the center of the middle interval, the interpolant and its derivative simplify to

$$y(x_{i+1/2}) = -\frac{1}{16}y_{i-1} + \frac{9}{16}y_i + \frac{9}{16}y_{i+1} - \frac{1}{16}y_{i+2}$$

$$y'(x_{i+1/2}) = \frac{1}{x_{i+1} - x_i}\left(\frac{1}{24}y_{i-1} - \frac{9}{8}y_i + \frac{9}{8}y_{i+1} - \frac{1}{24}y_{i+2}\right)$$

Cubic Lagrange methods

Analytic error performance

- Again re-derive the method by Taylor expansions, using $x_{\{i+2,i-1\}} = x \pm 3h$

$$f(x \pm h) = f(x) \pm hf'(x) + \frac{1}{2}h^2 f''(x) \pm \frac{1}{6}h^3 f'''(x) + \mathcal{O}(h^4)$$

$$f(x \pm 3h) = f(x) \pm 3hf'(x) + \frac{9}{2}h^2 f''(x) \pm \frac{9}{2}h^3 f'''(x) + \mathcal{O}(h^4)$$

- Then,

$$f(x) = \frac{9}{16}\Big(f(x+h) + f(x-h)\Big) - \frac{1}{16}\Big(f(x+3h) - f(x-3h)\Big) + \frac{3}{8}h^4 f^{(4)}(x)$$

$$f'(x) = \frac{9}{16h}\Big(f(x+h) - f(x-h)\Big) - \frac{1}{48h}\Big(f(x+3h) - f(x-3h)\Big) + \frac{18}{5}h^4 f^{(5)}(x)$$

Cubic spline methods

Method overview

- We could instead desire a function which is fully continuous and differentiable
- For each subinterval $[x_j, x_{j+1}]$, define the cubic

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^4$$

- Require continuity: $S_j(x_{j+1}) = S_{j+1}(x_{j+1})$
- Require twice differentiable: $S_j'(x_{j+1}) = S_{j+1}'(x_{j+1})$, $S_j''(x_{j+1}) = S_{j+1}''(x_{j+1})$
- Impose “natural” boundary conditions: $S''(x_0) = S''(x_n) = 0$

Cubic spline methods

Method overview

- Notice that at $x = x_j$, $S_j(x_j) = a_j$, $S_j'(x_j) = b_j$, and $S_j''(x_j) = c_j$
- Continuity constraints yield system of equations

$h_j \equiv x_{j+1} - x_j$

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(y_{j+1} - y_j) - \frac{3}{h_{j-1}}(y_j - y_{j-1})$$

$$b_j = \frac{1}{h_j}(y_{j+1} - y_j) - \frac{h_j}{3}(2c_j + c_{j+1}), \quad d_j = \frac{1}{3h_j}(c_{j+1} - c_j)$$

- Can solve for all values of c_j by using boundary conditions, propagate through to get the values of the b_j and d_j

Cubic spline methods

Analytic error performance

- Final interpolant is $S(x) = S_j(x)$ for $x_j \leq x \leq x_{j+1}$
- Derivative given by $S'(x)$
- By Quateroni et al., we have

$$\max |f(x) - S(x)| \leq \frac{5}{384} h^4 \max |f(x)|$$

$$\max |f'(x) - S'(x)| \leq \frac{1}{24} h^3 \max |f'(x)|$$

- Thus, the spline converges uniformly to the true function as the interval size (h) decreases and the number of sampling points increases
- Further, S is accurate up to cubic order in h, S' up to quadratic order

Forced-positive interpolation

- One feature of the distribution function f is that it must be positive
- We can build this into our interpolation scheme
- Instead of interpolate(x, y), we use

$$\exp[\text{interpolate}(x, \log y)]$$

- This guarantees positivity and gives a sizable accuracy boost for points close to zero

Numerical tests

Method

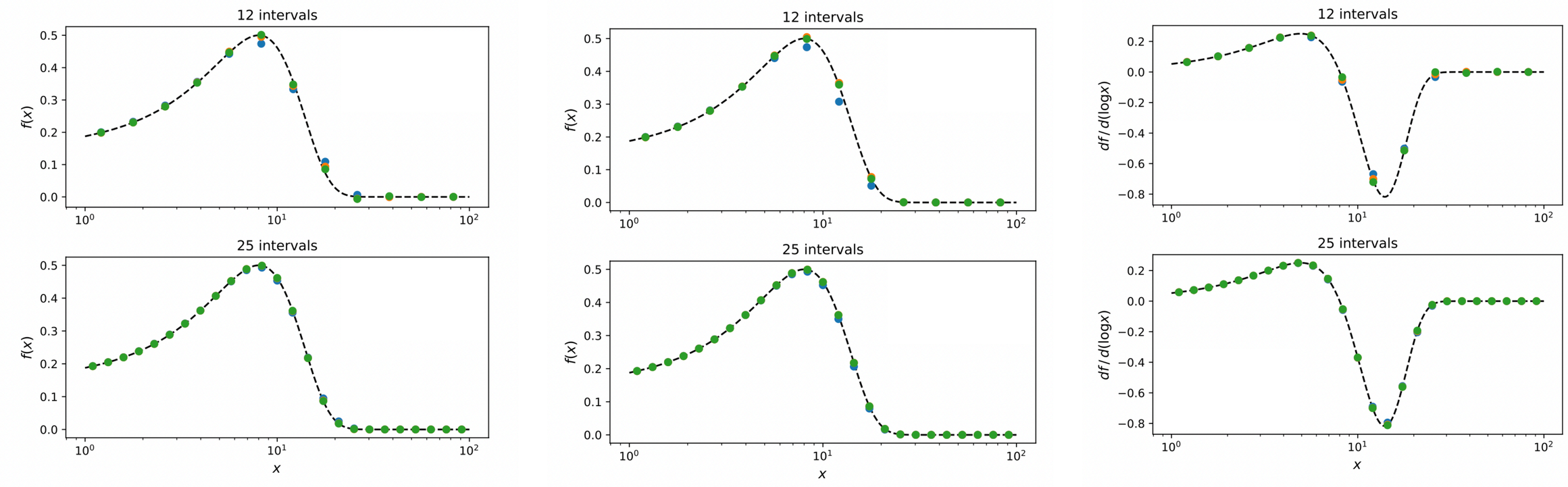
- To test these methods on a relatively physical example, we will take a Gaussian distribution

$$f(x) = \frac{1}{2} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

- We sample it at $n + 1$ logarithmically spaced points between 1 and 100
- We then apply the interpolation and differentiation techniques on $\log x$ to obtain estimates of f and $df/d(\log x)$ on the geometric midpoints of the sampling intervals

Numerical tests

Gaussian with $\mu = 8$ and $\sigma = 5$



Standard interpolation

Forced-positive interpolation

Differentiation

Numerical tests

Error performance

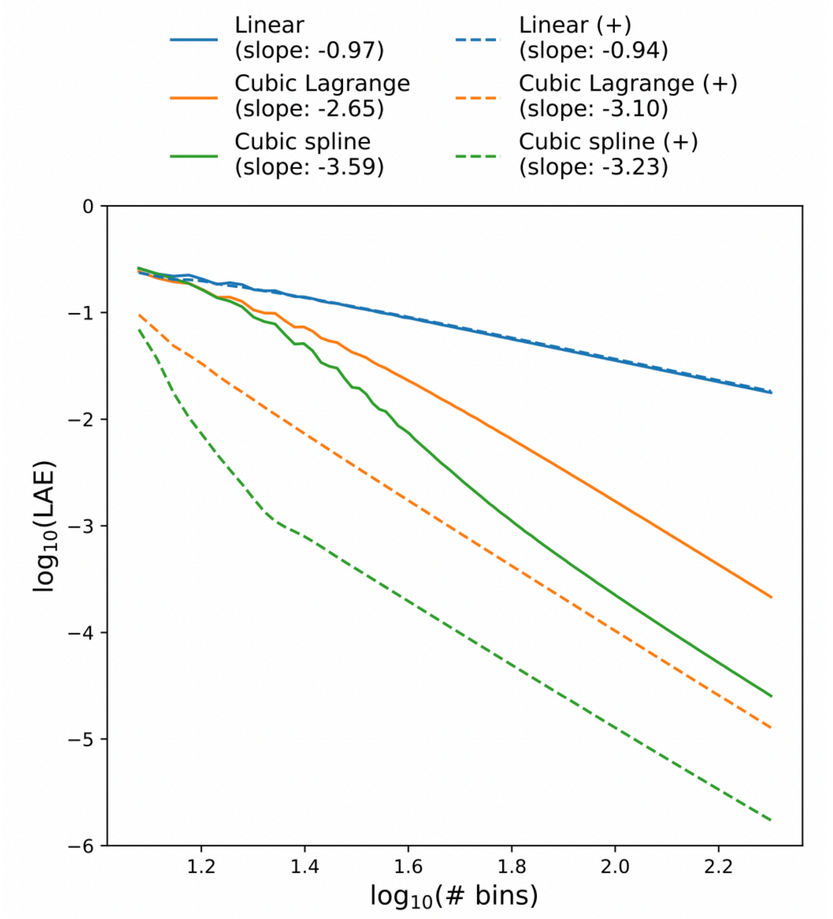
- We can describe the error of these methods quantitatively
- Use the **least absolute error** (LAE)

$$\text{LAE} = \sum_{i=0}^{n-1} \left| f_i - f(x_{i+1/2}) \right|$$

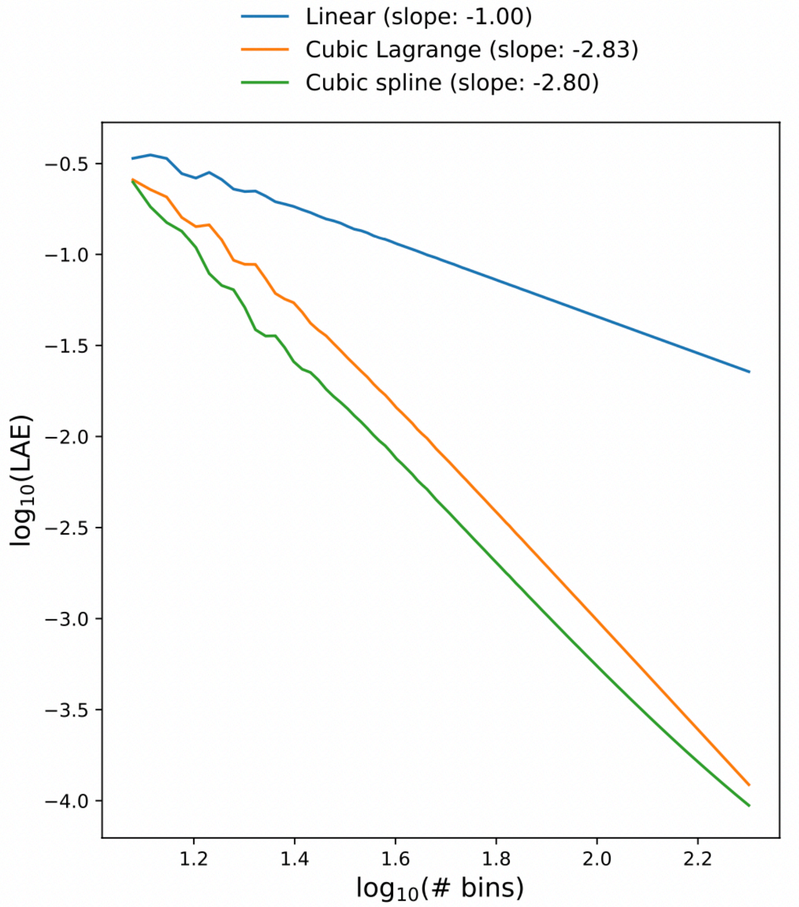
- We assume LAE $\propto n^k$ for some k
  - On a plot of $\log(\text{LAE})$ vs $\log(n)$, slope is k

Numerical tests

Error



Interpolation
(Forced-positive with the “(+)”)

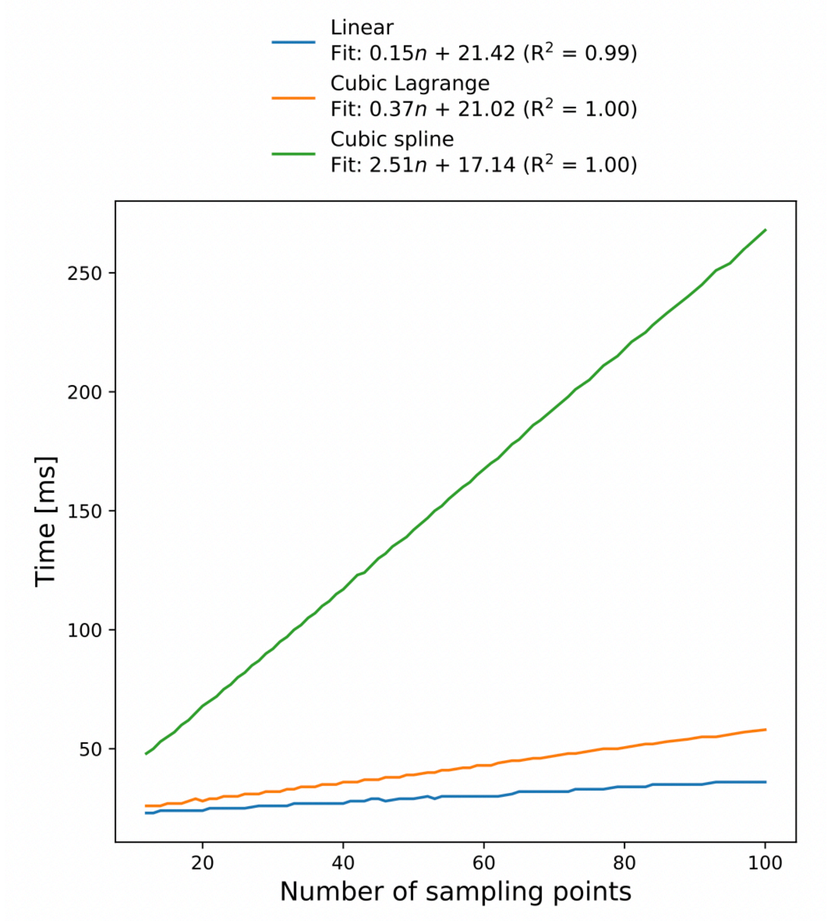


Differentiation

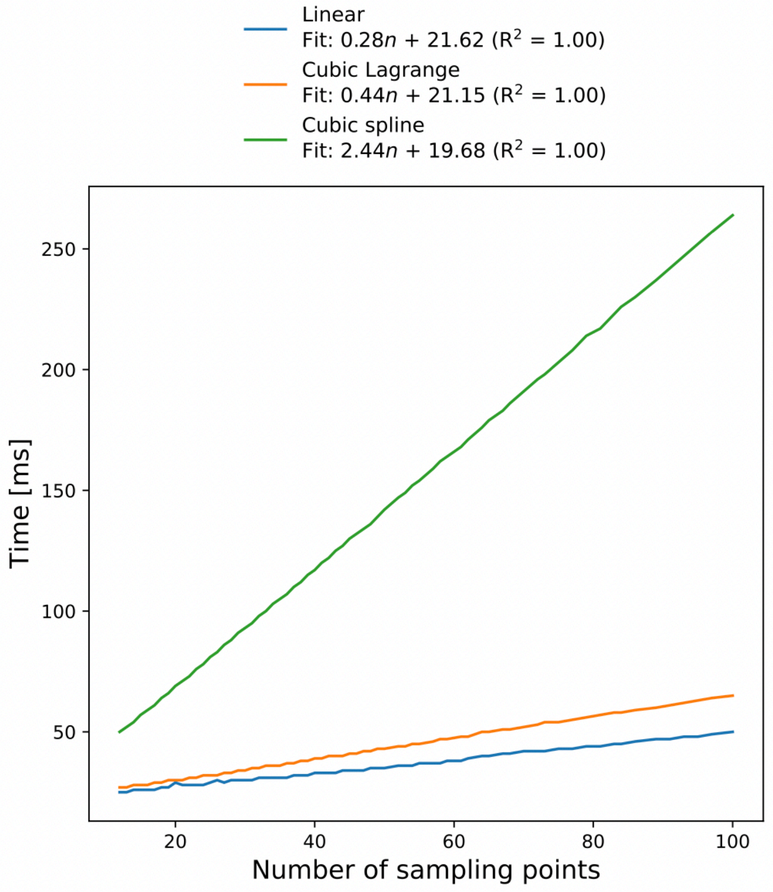
- 50 trials per data point with Gaussians of means between 5 and 25
- 12 to 200 bins

Numerical tests

Runtime



Interpolation



Differentiation

- 1000 trials per data point
- Differentiation slower than interpolation (except for spline, where actually *less* work is needed)

Integration
Euler's method

- Suppose we have an ODE $dy/dt = f(t, y)$ at time t_n and want to take a step of size h to time t_{n+1}
- Expand $y(t_{n+1})$ about t_n

$$y(t_{n+1}) = y(t_n) + h \left.\frac{dy}{dt}\right|_{t=t_n} + \mathcal{O}(h^2)$$

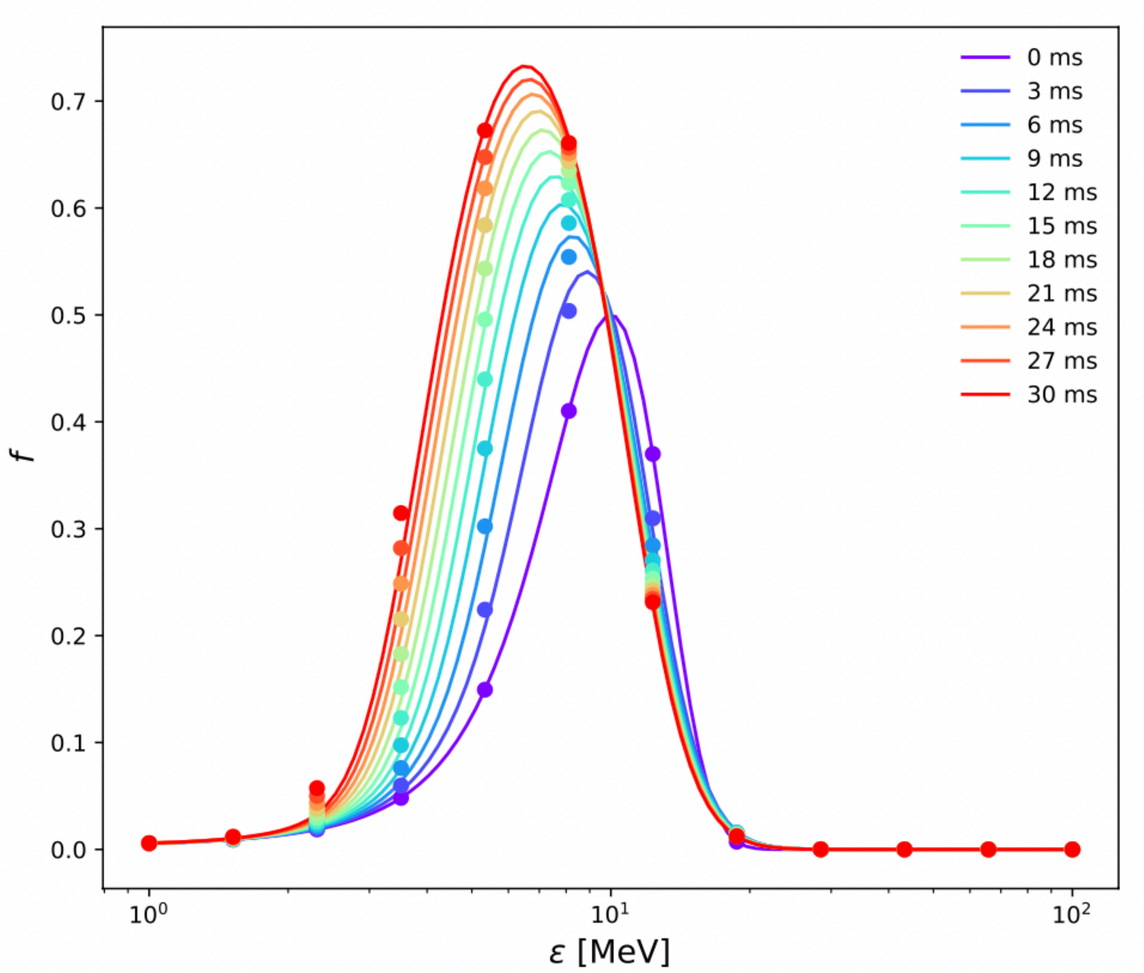
- Plug in ODE

$$y_{n+1} = y_n + hf(t_n, y_n) + \mathcal{O}(h^2)$$

Full solver

Simple results

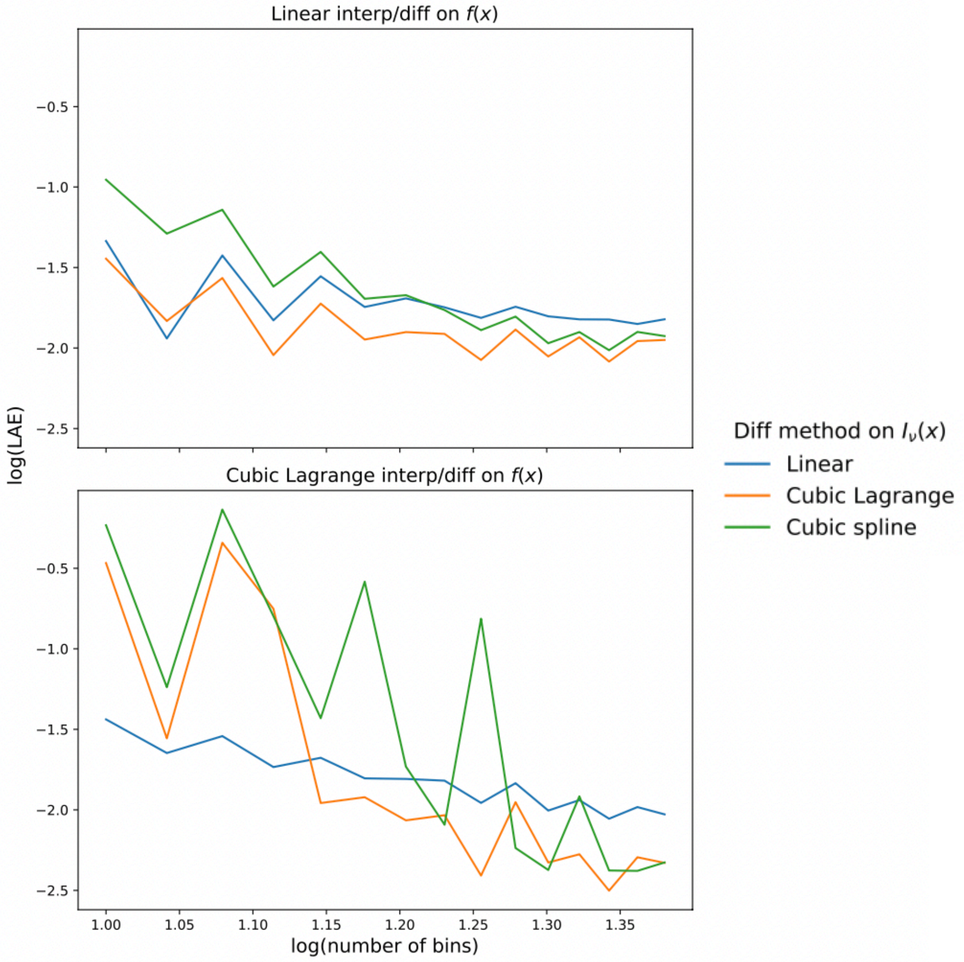
- Start with Gaussian ($\mu = 10, \sigma = 3$)
- Use fixed $dt = 1$ μs, 100 and 12 energy bins
- Integrate forward 30,000 steps
- $kT = 2$ MeV, $\rho = 10^{11}$ g/cm^3, $Y_e = 0.2$



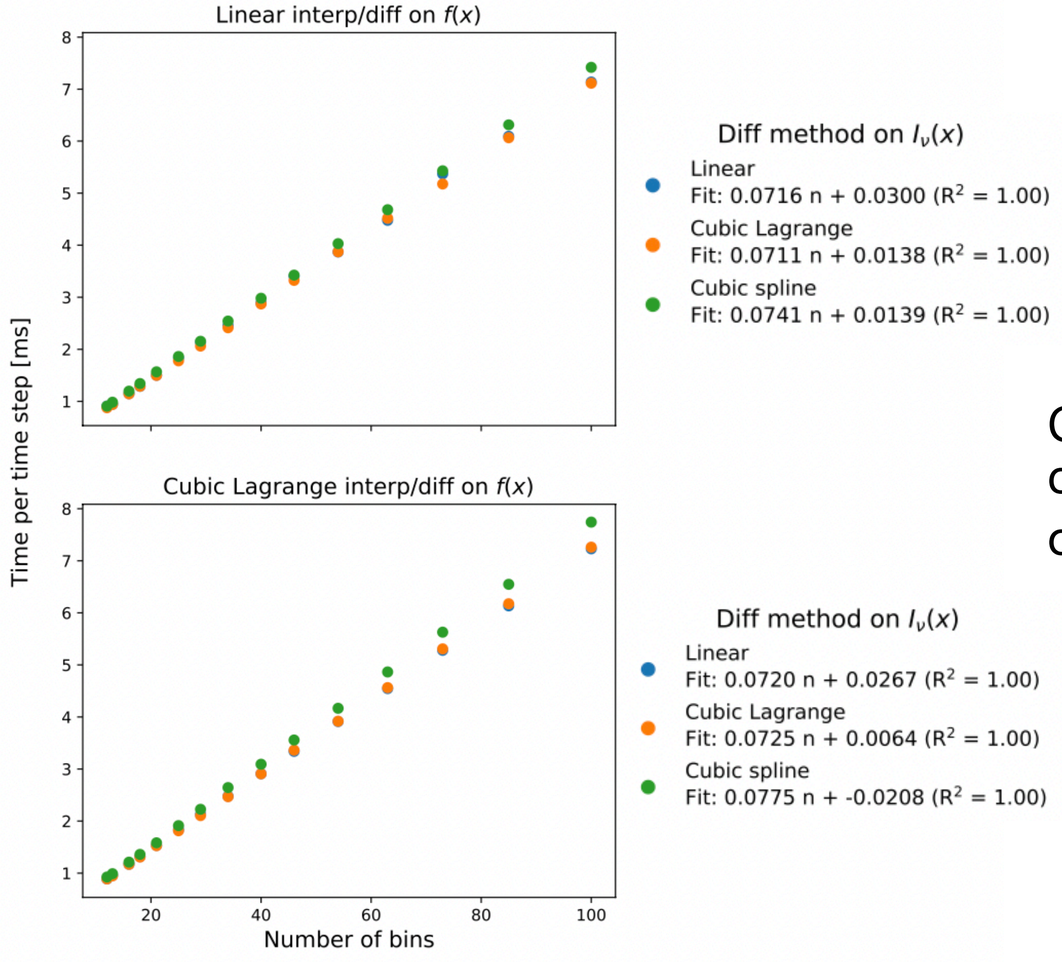
Solid line: 100 bins, circles: 12 bins

Full solver

Performance characterization



Least absolute error (compared to a 100 bin run)



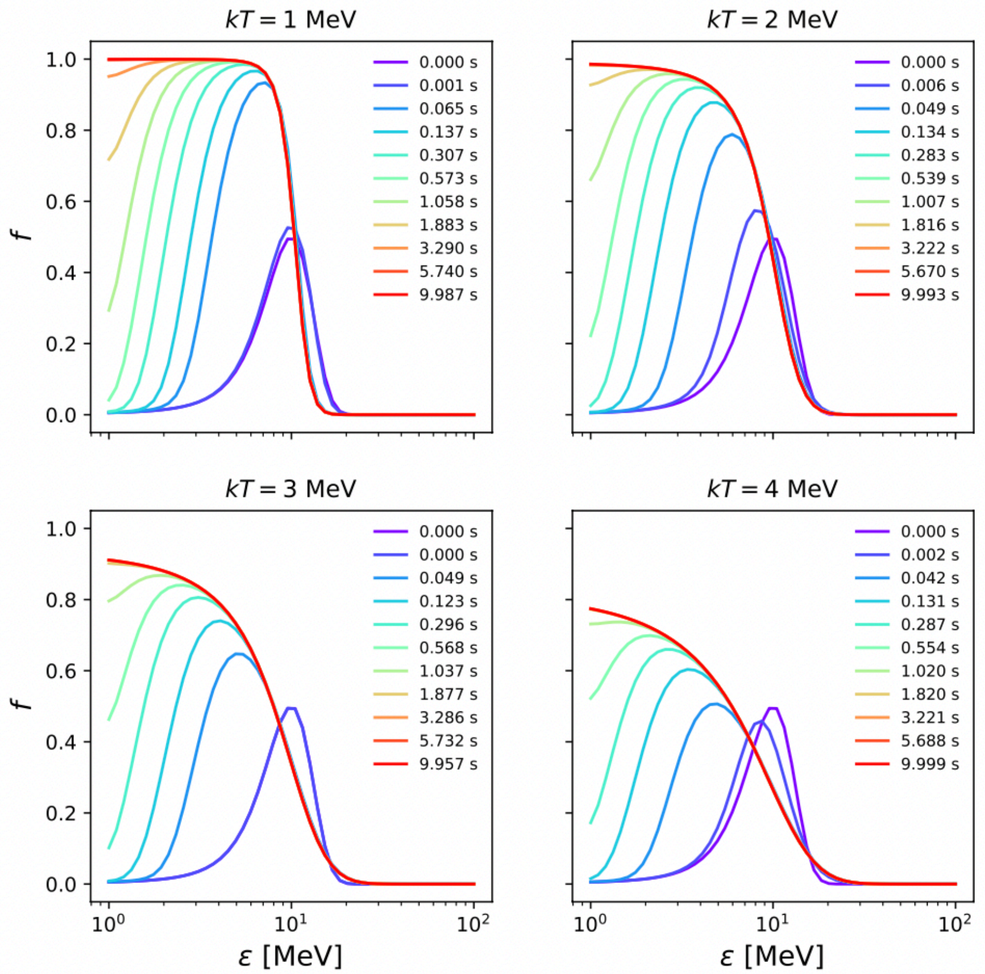
Runtime

Choose to use cubic Lagrange on f, linear on I_ν

Full solver

Late-time results for varying temperatures

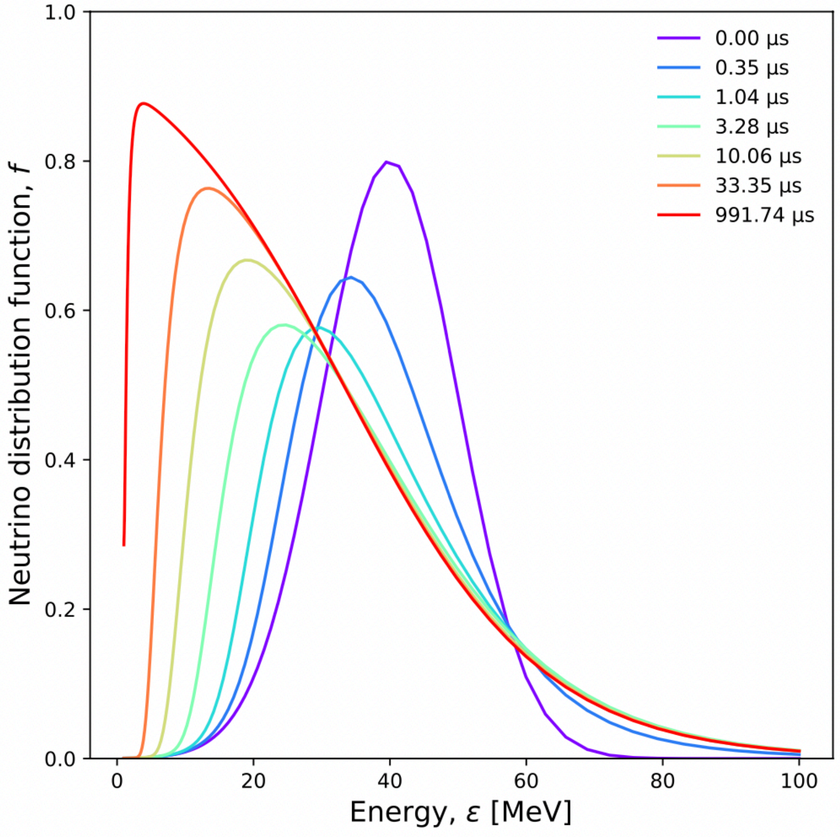
- Expect shape of distribution to change dramatically for varying temperature kT
- Final distribution described by Fermi-Dirac distribution
- High kT limit yields Maxwell-Boltzmann distribution



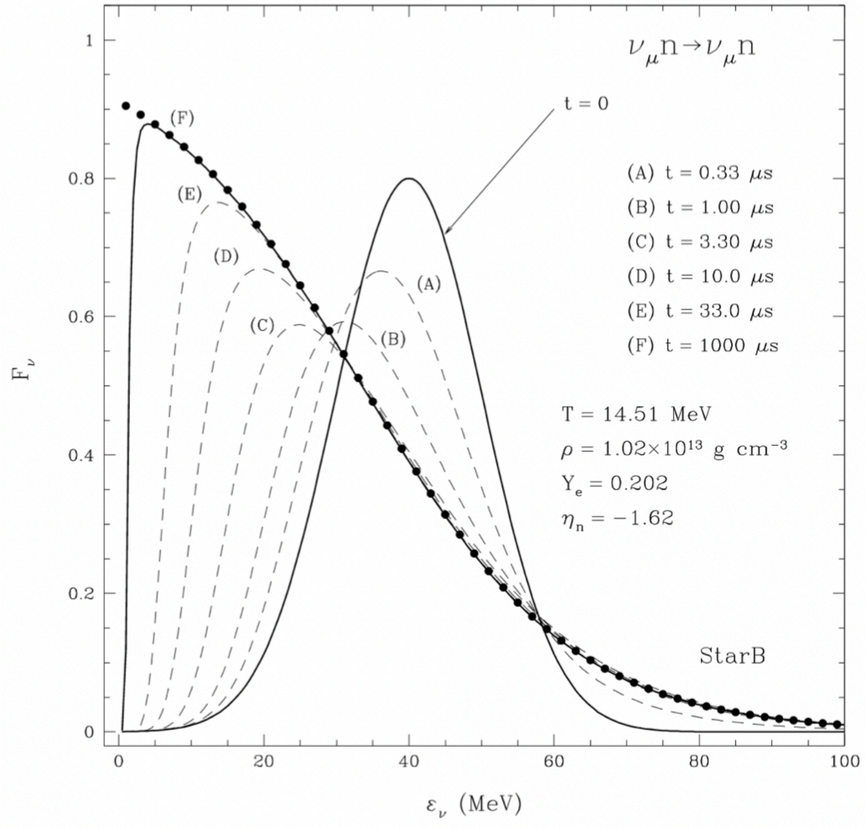
Full solver

Comparison to past work

- Gaussian with height 0.8, $\mu = 40, \sigma = 10$
- Use 100 bins
- $kT = 14.51$ MeV, $\rho = 1.02 \times 10^{13}$ g/cm^3, $Y_e = 0.202$
- Neutron interactions only



Our solver

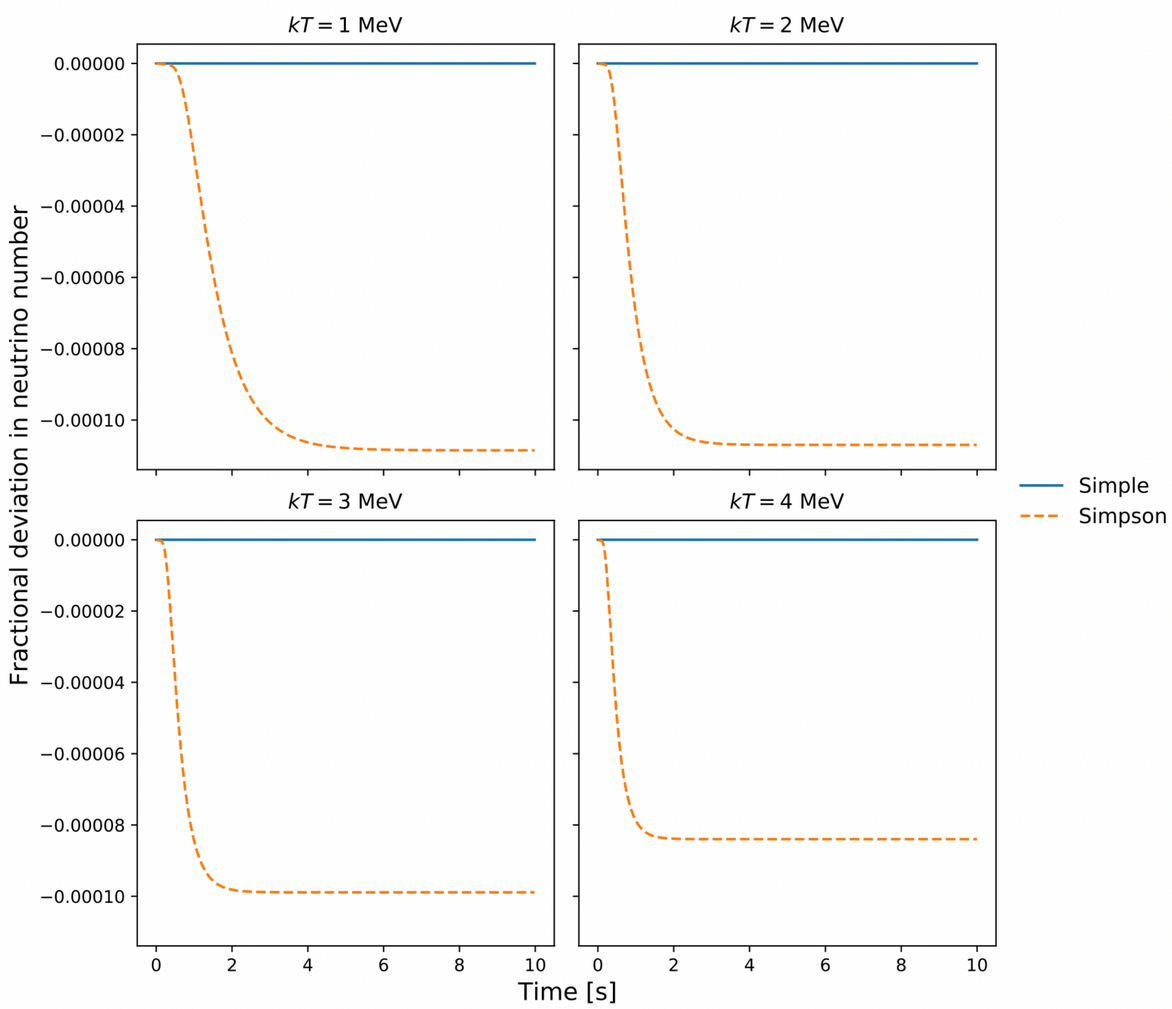


Thompson et al., 2000

Full solver

Neutrino conservation

- Integrate $x^2 f$ over all energy bins
- Conservation guaranteed with simple integral
- Higher-order Simpson's method shows that it is conserved to within 0.01%



Conclusions

- While perhaps the most accurate and generally applicable interpolation method, cubic spline methods are very slow
- Cubic Lagrange methods are much faster, but can struggle for very small numbers of bins
- Using linear methods helps improve stability in the solver for low bin counts
- The resulting solver conserves neutrino number, is highly accurate even to low numbers of bins, and agrees with past work

Acknowledgments

- I’d like to thank Prof. Adam Burrows for allowing me to work on a very interesting research topic, for his insightful suggestions, and most of all for taking the time to advise me over the past semester

- I’d also like to thank Prof. Ramon van Handel for his very insightful advice last semester when I was struggling to find a research topic, as well as Tianshu Wang for his advice on some details of the solver’s implementation

Backup

Linear methods

Method overview

- Given two points (x_i, y_i) and (x_{i+1}, y_{i+1}), we can linearly interpolate

$$y(x) = \frac{x - x_{i+1}}{x_i - x_{i+1}} y_i + \frac{x - x_i}{x_{i+1} - x_i} y_{i+1}$$

- Differentiating yields

$$y'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

- At the interval midpoint, these become

$$y(x_{i+1/2}) = \frac{1}{2}\left(y_i + y_{i+1}\right), \quad y'(x_{i+1/2}) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Cubic Lagrange methods

Method overview

- We obtained our linear methods through linear Lagrange interpolation
- Let's increase this to cubic → consider now four points $(x, y)_{\{i-1,i,i+1,i+2\}}$

$$f(x) \approx \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}y_0 + \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)}y_1$$
$$+\frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}y_3$$

Cubic Lagrange methods

Method overview

$$
\begin{aligned}
f'(x) \approx & \frac{(x-x_1)(x-x_2)+(x-x_1)(x-x_3)+(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} y_0 \\
& + \frac{(x-x_0)(x-x_2)+(x-x_0)(x-x_3)+(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} y_1 \\
& + \frac{(x-x_0)(x-x_1)+(x-x_0)(x-x_3)+(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} y_2 \\
& + \frac{(x-x_0)(x-x_1)+(x-x_0)(x-x_2)+(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} y_3
\end{aligned}
$$

Cubic Lagrange methods

Method overview

- We can make use of the fact that the bins are equispaced and that we desire x at the center of the middle interval
- Previous equations simplify to

$$y(x_{i+1/2}) = -\frac{1}{16}y_{i-1} + \frac{9}{16}y_i + \frac{9}{16}y_{i+1} - \frac{1}{16}y_{i+2}$$

$$y'(x_{i+1/2}) = \frac{1}{x_{i+1} - x_i}\left(\frac{1}{24}y_{i-1} - \frac{9}{8}y_i + \frac{9}{8}y_{i+1} - \frac{1}{24}y_{i+2}\right)$$

Integration
Runge-Kutta

- Higher order stepping methods
- Second-order:
  - Take a trial step to the midpoint of the time interval
  - Compute dy/dt at the midpoint
  - Use this derivative to go back to take the full step from the beginning

$$k_1 = hf(t_n, y_n)$$

$$k_2 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1)$$

$$y_{n+1} = y_n + k_2 + \mathcal{O}(h^3)$$

Integration
Runge-Kutta

- Fourth-order:
  - Compute the derivative four times: t_n, $t_n + h/2$ (twice), and $t_n + h$
  - Combine derivatives to cancel out various orders in h

$$
\begin{aligned}
k_1 &= h f(t_n, y_n) \\
k_2 &= h f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1) \\
k_3 &= h f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_2) \\
k_4 &= h f(t_n + h, y_n + k_3) \\
y_{n+1} &= y_n + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(h^5)
\end{aligned}
$$

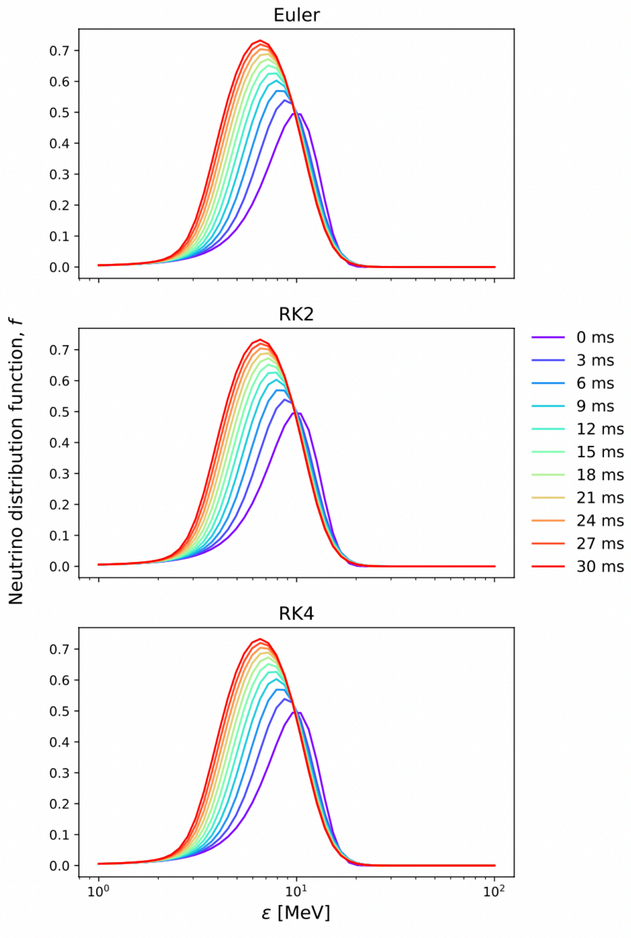
Integration

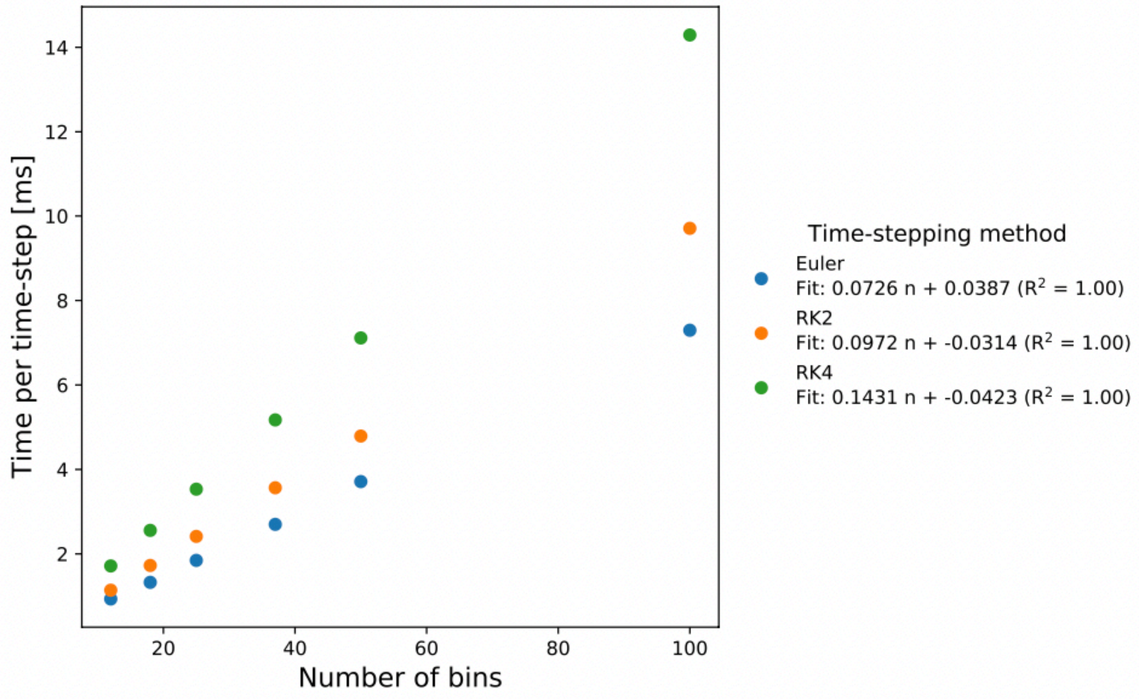
Step-size control

- Empirically, we found that more important than the stepper was the step-size control
- We thus implement a rudimentary adaptive step size
- Set a desired fractional change ξ and a maximum allowed change dt_m
- Maximum fractional change to the distribution after one step is $\max(\Delta y/y)$
- Target step-size: $dt' = dt\dfrac{\xi}{\max(\Delta y/y)}$
- Update step size to dt' if in $[dt - dt_m, dt + dt_m]$, otherwise set to corresponding interval limit

Full solver

ODE steppers: identical results, differing runtimes



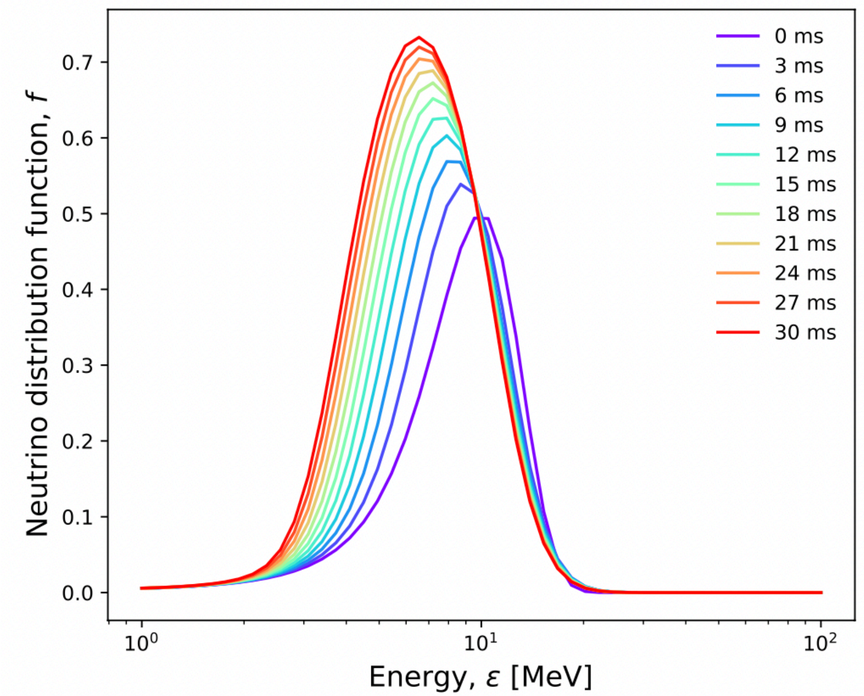


Best just to use Euler's method for our purposes!

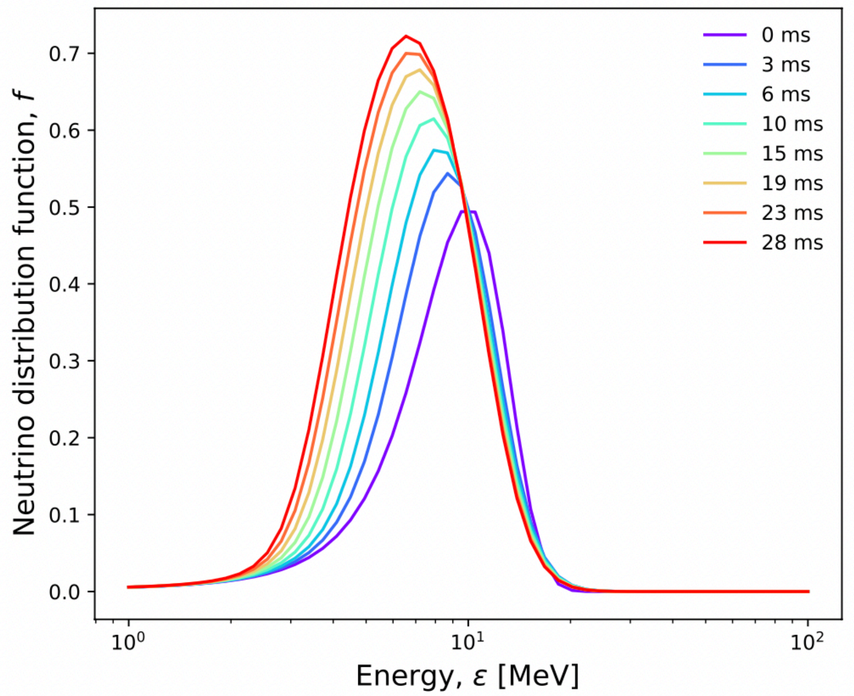
Full solver

Fixed vs adaptive step-size

- Integrate 0.03 seconds using 50 bins
- Starting distribution: Gaussian with $\mu = 10$ and $\sigma = 3$
- Compare fixed 1 μs steps with adaptive step size



Fixed



Adaptive