PDF Evolution in *x*-space

Daniel Adamiak

With contributions from Chris Cocuzza, Kevin Braga, Nobuo Sato

Basics of PDF Phenomenology

Experiments can measure structure functions (like F2 in DIS) that depend of PDFs:

$$F_2(x, Q^2) = x \sum_q e_q^2 \int_x^1 \frac{dz}{z} \, C_q(x/z, \alpha_s) f_q(x, Q^2)$$

The PDFs are computed via the DGLAP evolution equation:

$$\frac{\partial}{\partial \ln Q^2} f_i(x, Q^2) = \frac{\alpha_s}{2\pi} \int_x^1 \frac{dz}{z} p_{ij}(x/z, Q^2) f_i(x, Q^2)$$

Assuming a boundary condition of the form:

$$f_i(x, \mu^2) = N_i x^{a_i} (1 - x)^{b_i} (\ldots)$$

This convolution

$$\frac{\partial}{\partial \ln Q^2} f_i(x, Q^2) = \frac{\alpha_s}{2\pi} \int_x^1 \frac{dz}{z} p_{ij}(x/z, Q^2) f_i(x, Q^2)$$

Can be very slow, especially when needed for multiple Q values and different initial conditions. If only there was a way to separate the evolution from the initial condition…

Easier in Mellin Space

$$F(N) = \int_0^\infty dx x^{N-1} f(x)$$

Mellin Transform

$$f(x) = \frac{1}{2\pi i} \int_c dN x^{-N} F(N)$$

Inverse-Mellin Transform

$$\int_x^1 \frac{dz}{z} g(x/z) f(z) \to G(N) F(N)$$

Convolutions become much simpler

The evolution convolution is now separable!

$$\int_x^1 \frac{dz}{z} p_{ij}(x/z) f_j(z) \rightarrow P_{ij}(N) F_j(N)$$

The evolution kernel, *P*, can now be precomputed. Evolution takes as long to compute as it does to multiply the two functions.

Evolution is very quick in Mellin space

Numerical implementation Of Inverse Mellin-Transform

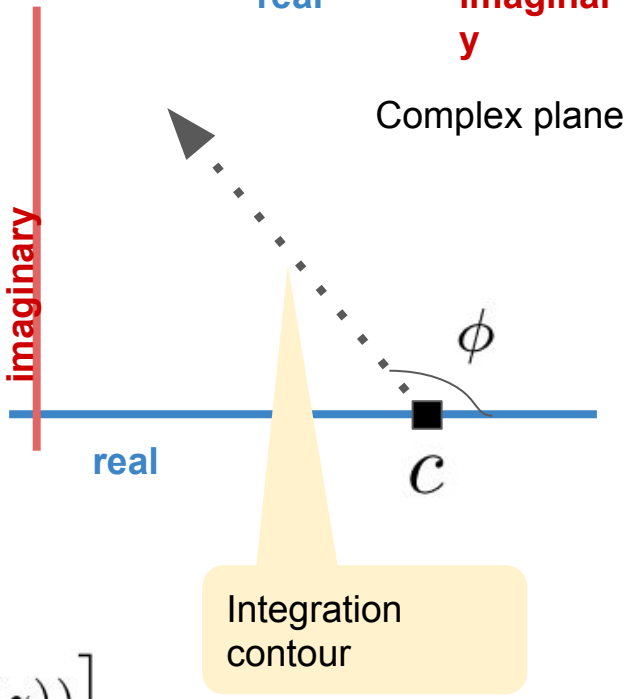
$$f(x) = \frac{1}{2\pi i}\int_c dN x^{-N} F(N)$$

$$N(z) = c + ze^{i\phi}$$

$$f(x) = \frac{1}{\pi}\int_0^\infty dz \mathrm{Im}\left[e^{i\phi}x^{-N(z)}F(N(z))\right]$$

$$N(z) = [c + z\cos(\phi)] + i[z\sin(\phi)]$$

real imaginary



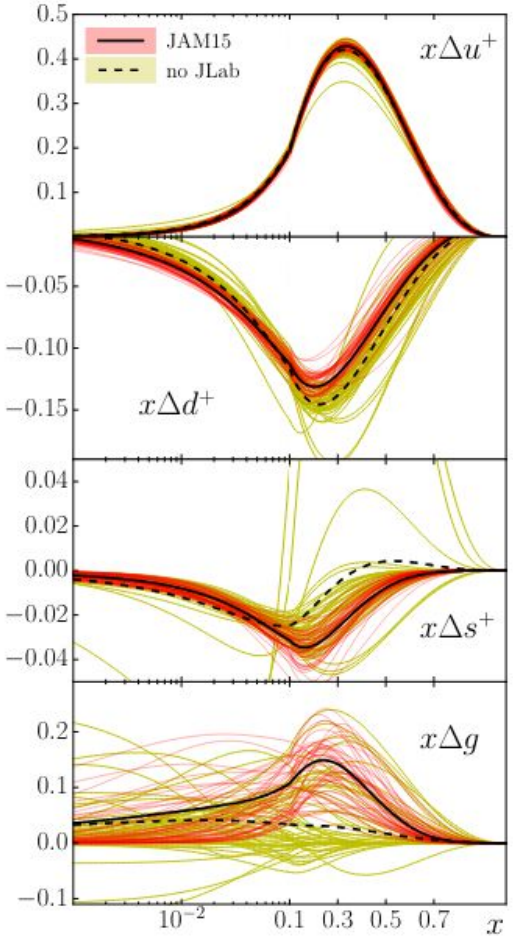
This ‘pole condition’ limits the class of initial conditions we can Mellin transform. This is part of the reason the parametric form is so appealing

$$f(x,\mu^2) = Nx^a(1-x)^b \rightarrow \frac{N\Gamma(N+a)\Gamma(b+1)}{\Gamma(N+a+b+1)}$$

There’s no guarantee that we can understand the pole structure if we use a more ‘universal’ initial condition

Find parameters that best describe the data. Many such choices lead to many ‘replicas’. The distribution of the replicas give us the uncertainty of the parameterized model

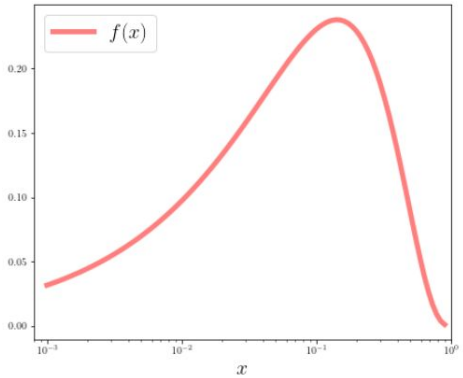
Nobuo Sato, WM, Sebastian Kuhn, Jake Ethier, Alberto Accardi: Phys. Rev. D 93, 074005 (2016)

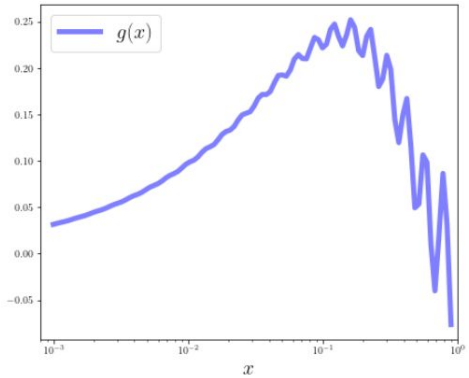


Is this the only model that matches the data?

Parameterized models lead to artificial certainty, especially at large *x*.

We explore models that look like this





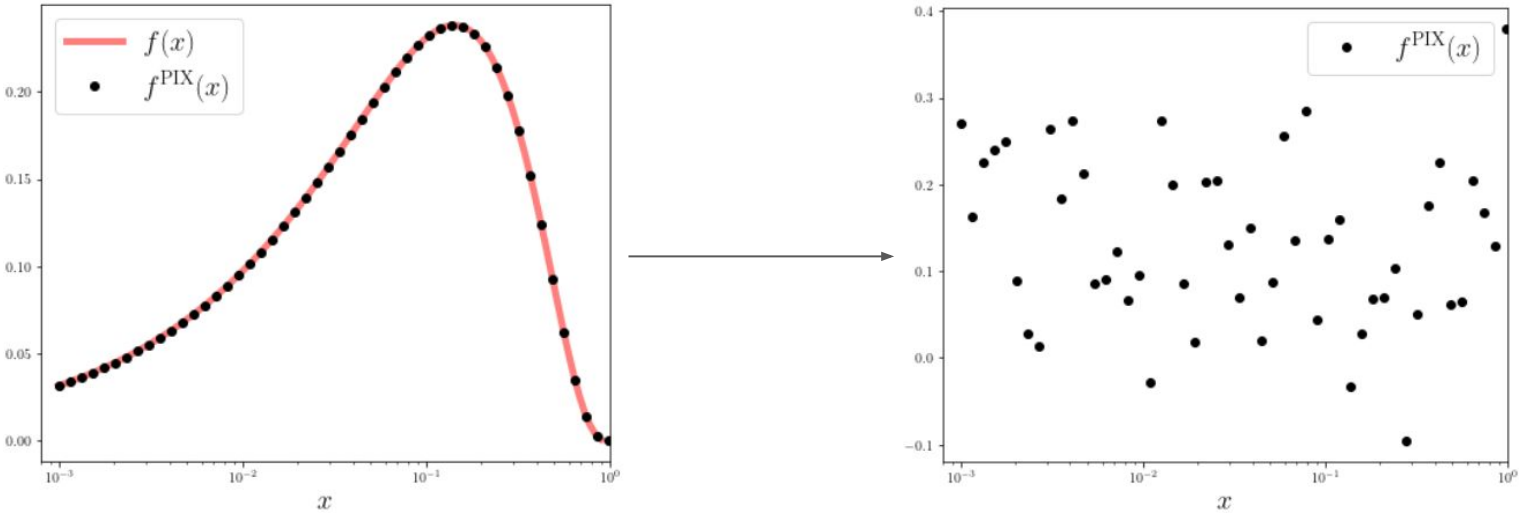
Not like this

How might we explore the impact data has on input scale PDF without having to worry about model bias?

We need a universal function - test all models simultaneously

Pixelized PDFs

Instead of a parametric model, we ‘pixelize’ our input scale PDF. Each pixel is its own free parameter.



This is a universal function approximator, at the cost (benefit?) of introducing a finite resolution in *x*.

Can’t Pixelize in Mellin Space

We need analytic functions if we want to go to Mellin space

But the pixelized PDF is a numeric function.

We are forced to do evolution in *x*-space. Can we design a faster evolution algorithm? Can we ‘separate’ the evolution kernel from the initial condition?

Evolution is differentiation

$$f(x, t + \Delta t) \approx f(x, t) + \Delta t \partial_t f(x, t)$$

Differentiation is integration (DGLAP)

$$\partial_t f(x, t) = (K \otimes f)(x, t)$$

Integration is matrix multiplication (Linearity)

$$(K \otimes f)(x, t) \approx K_{ij}(t) f_j(t)$$

Evolution is matrix multiplication

$$f_i(t + \Delta t) \approx (\delta_{ij} + \Delta t K_{ij}(t)) f_j(t)$$

$$\partial_t f_i(t) \approx K_{ij}(t) f_j(t)$$

Iterate to build up Evolution

$$f_i(t+2\Delta t) \approx \left(\delta_{ij} + \Delta t K_{ij}(t+\Delta t)\right)\left(\delta_{jk} + \Delta t K_{jk}(t)\right) f_k(t)$$

$$f_i(t+2\Delta t) \approx S_{ij}(t+\Delta t) S_{jk}(t) f_k(t)$$

$$f_i(t+N\Delta t) \approx E_{ij}(t+N\Delta t) f_j(t) = \left(\prod_n K(t+n\Delta t)\right)_{ij} f_j(t)$$

Complete separation of evolution and initial condition. Matrix multiplication is very fast on GPUs

Integration is Linear

Example: Riemann sums

$$\int_x^1 dz F(z) \approx \sum_{j=i}^{N} (x_j - x_{j-1}) F(x_j) = R_{ij} F_j$$

Riemann sum approximation

Where $R_{ij} = \begin{pmatrix} 0 & \Delta_1 & \Delta_2 & \Delta_3 & \dots \\ 0 & 0 & \Delta_2 & \Delta_3 & \dots \\ 0 & 0 & 0 & \Delta_3 & \dots \\ \vdots & \vdots & \ddots & \ddots & \dots \end{pmatrix}$, $\Delta_j = x_j - x_{j-1}$

More details on the Kernels

DGLAP splitting kernel contains ‘plus’ distributions

$$\int_x^1 \frac{dz}{z} \frac{p(x/z)f(z)}{(1-z)_+} = \int_x^1 dz \left[\frac{1}{1-z}\left(\frac{1}{z}p(x/z)f(z) - p(x)f(1)\right)\right] + p(x)f(1)\ln(1-x)$$

$K_z(x,z)$ $K_x(x,z)$ $K_c(x)$

Also includes non-plus distribution parts. Requires full matrix integration

Returns diagonal matrix. Integration is independent of *f*, hence independent of grid

Proportional to Identity matrix - no integration

Improving Matrix Evolution

- Go from Euler’s method to Runge-Kuta
- Instead of Riemann sums, use Gaussian Quadrature
  - This will require interpolation
- Intelligent choice of grid

Runge-Kuta

$$
\begin{aligned}
f_i(t+dt) &= dE_{ij}(t) f_j(t) \\
&= \delta_{ij} f_j(t) \\
&\quad + \frac{dt}{6}\left[\partial_{ij}(t) + 4\partial_{ij}\left(t+\frac{dt}{2}\right) + \partial_{ij}(t+dt)\right] f_j(t) \\
&\quad + \frac{dt^2}{6}\left[\partial_{ik}\left(t+\frac{dt}{2}\right)\partial_{kj}(t) + \partial_{ik}\left(t+\frac{dt}{2}\right)\partial_{kj}\left(t+\frac{dt}{2}\right) + \partial_{ik}(t+dt)\partial_{kj}\left(t+\frac{dt}{2}\right)\right] f_j(t) \\
&\quad + \frac{dt^3}{12}\left[\partial_{ik}\left(t+\frac{dt}{2}\right)\partial_{kl}\left(t+\frac{dt}{2}\right)\partial_{lj}(t) + \partial_{ik}(t+dt)\partial_{kl}\left(t+\frac{dt}{2}\right)\partial_{lj}\left(t+\frac{dt}{2}\right)\right] f_j(t) \\
&\quad + \frac{dt^4}{24}\partial_{ik}(t+dt)\partial_{kl}\left(t+\frac{dt}{2}\right)\partial_{lm}\left(t+\frac{dt}{2}\right)\partial_{mj}(t) f_j(t)
\end{aligned}
$$

Improved Integration Scheme: Gaussian Quadrature

$$\int_x^1 dz F(z) \approx \sum_g w_g J_{jg} F(x_g)$$

- Gaussian weights: w_g
- Roots of Legendre polynomials: x_g
- Jacobian: J_{jg}

The Gauss grid is not the most natural - we might prefer to interpolate onto it

Gaussian Quadrature Continued:

$$K^{ij} = \sum_g w^g J^{ig} \left(K_z^{ig} L^{gj} + K_x^{ig} \delta^{ij} \right) + K_c^j \delta^{ij}$$

- With interpolation matrix L^{gj}

Interpolation is also a Matrix

Example: Linear interpolation

$$f(x_l) = \frac{x_{i+1} - x_l}{x_{i+1} - x_i} f(x_i) + \frac{x_l - x_i}{x_{i+1} - x_i} f(x_{i+1})$$

$$L_{li} = \frac{x_{i+1} - x_l}{x_{i+1} - x_i}$$

$$L_{li+1} = \frac{x_l - x_i}{x_{i+1} - x_i}$$

Interpolation is also a Matrix

Example: Modified Piecewise Hermite Cubic Polynomials

$$y(x) = A + B(x - x_L) + C(x - x_L)^2 + D(x - x_L)^2(x - x_R)$$

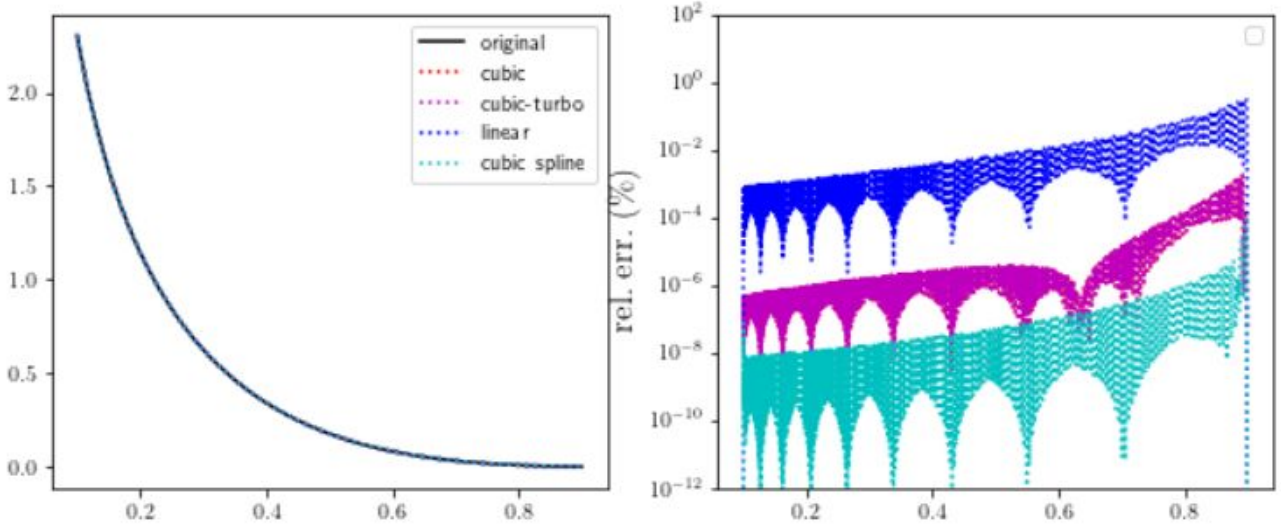
$$A = y_L$$
$$B = S_L$$
$$C = \frac{y' - S_L}{x_R - x_L}$$
$$D = \frac{S_L + S_R - 2y'}{(x_R - x_L)^2}$$
$$y' = \frac{y_R - y_L}{x_R - x_L}$$

$$S_L \approx \frac{y_R - y_{L-1}}{x_R - x_{L-1}}$$
$$S_R \approx \frac{y_{R+1} - y_L}{x_{R+1} - x_L}$$

- For endpoints, I just brute a cubic polynomial

Solve for *yR* and *yL* to obtain interpolation matrix

Comparing accuracy of different interpolation methods



- Cubic does much better than linear, 10e-4% relative error
- Still a gap between matrix interpolation and state-of-the-art cubic spline interpolation

Cubic Splines: Can we do better?

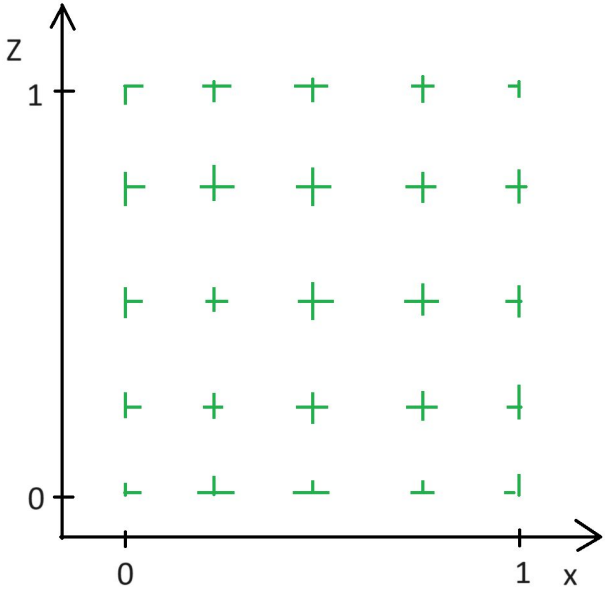
No, I can't

For a grid size of 5, a cubic spline interpolation matrix would be constructed out of the following rows:

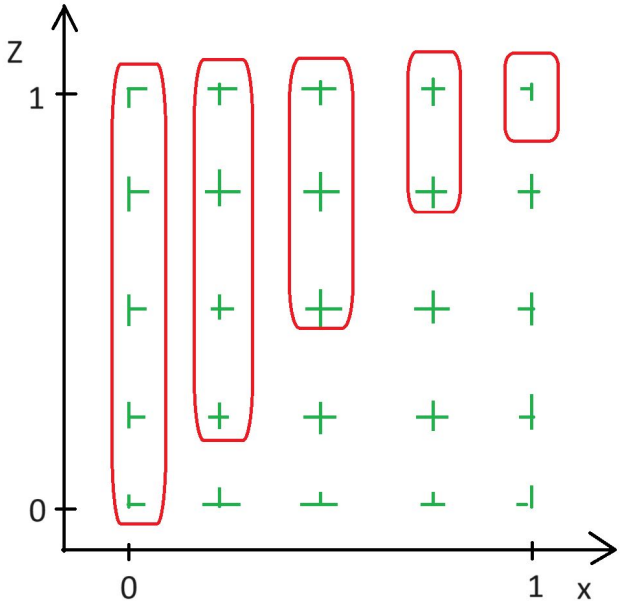
$$\begin{pmatrix} 1 + dx\left(-\frac{1}{2} - \frac{5\,h}{112}\right) + \frac{5\,dx^3}{112\,h} & dx\left(\frac{1}{2} + \frac{17\,h}{168}\right) - \frac{17\,dx^3}{168\,h} & \frac{dx^3}{14\,h} - \frac{dx\,h}{14} & -\frac{dx^3}{56\,h} + \frac{dx\,h}{56} & \frac{dx^3}{336\,h} - \frac{dx\,h}{336} \\ \frac{15\,dx^2}{112} - \frac{19\,dx^3}{336\,h} - \frac{13\,dx\,h}{168} & 1 - \frac{17\,dx^2}{56} + dx\left(-\frac{1}{2} + \frac{11\,h}{84}\right) + \frac{29\,dx^3}{168\,h} & \frac{3\,dx^2}{14} + dx\left(\frac{1}{2} - \frac{h}{42}\right) - \frac{4\,dx^3}{21\,h} & -\frac{3\,dx^2}{56} + \frac{5\,dx^3}{56\,h} - \frac{dx\,h}{28} & \frac{dx^2}{112} - \frac{5\,dx^3}{336\,h} + \frac{dx\,h}{168} \\ -\frac{dx^2}{28} + \frac{5\,dx^3}{336\,h} + \frac{dx\,h}{48} & \frac{3\,dx^2}{14} - \frac{5\,dx^3}{56\,h} - \frac{dx\,h}{8} & 1 - \frac{5\,dx^2}{14} + dx\left(-\frac{1}{2} + \frac{h}{6}\right) + \frac{4\,dx^3}{21\,h} & \frac{3\,dx^2}{14} + dx\left(\frac{1}{2} - \frac{h}{24}\right) - \frac{29\,dx^3}{168\,h} & -\frac{dx^2}{28} + \frac{19\,dx^3}{336\,h} - \frac{dx\,h}{48} \\ \frac{dx^2}{112} - \frac{dx^3}{336\,h} - \frac{dx\,h}{168} & -\frac{3\,dx^2}{56} + \frac{dx^3}{56\,h} + \frac{dx\,h}{28} & \frac{3\,dx^2}{14} - \frac{dx^3}{14\,h} - \frac{dx\,h}{7} & 1 - \frac{17\,dx^2}{56} + dx\left(-\frac{1}{2} + \frac{17\,h}{84}\right) + \frac{17\,dx^3}{168\,h} & \frac{15\,dx^2}{112} + dx\left(\frac{1}{2} - \frac{5\,h}{56}\right) - \frac{5\,dx^3}{112\,h} \end{pmatrix}$$

I can't predict how this changes as a function of grid size, *n.*

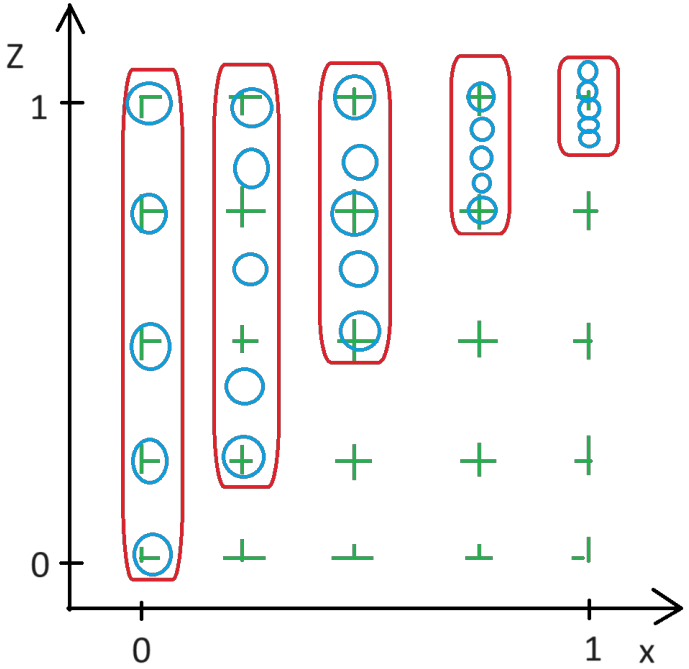
Intelligent choice of *x*-*z* grid



Intelligent choice of *x*-*z* grid

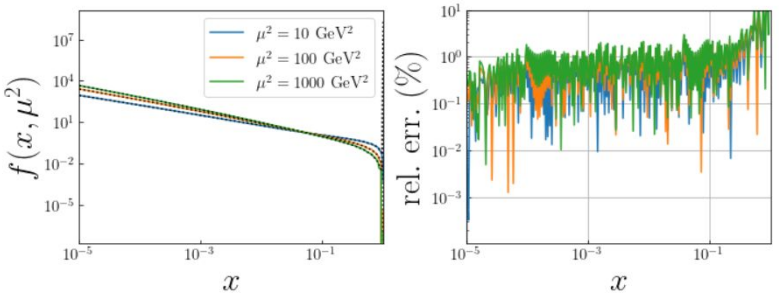


Intelligent choice of *x*-*z* grid

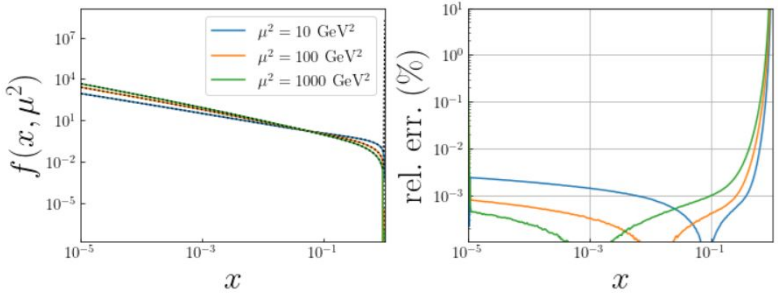


Sure, we interpolate f(x), but we don't have to interpolate the kernels

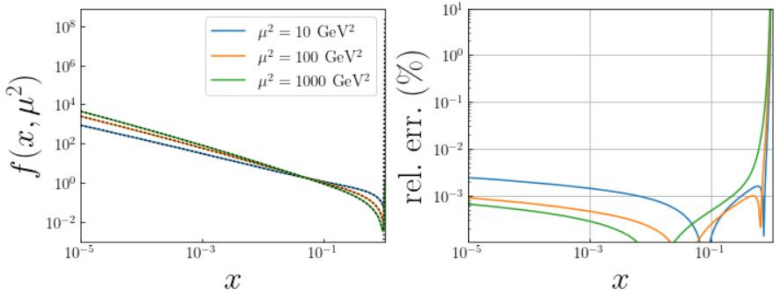
old Matrix



New Matrix

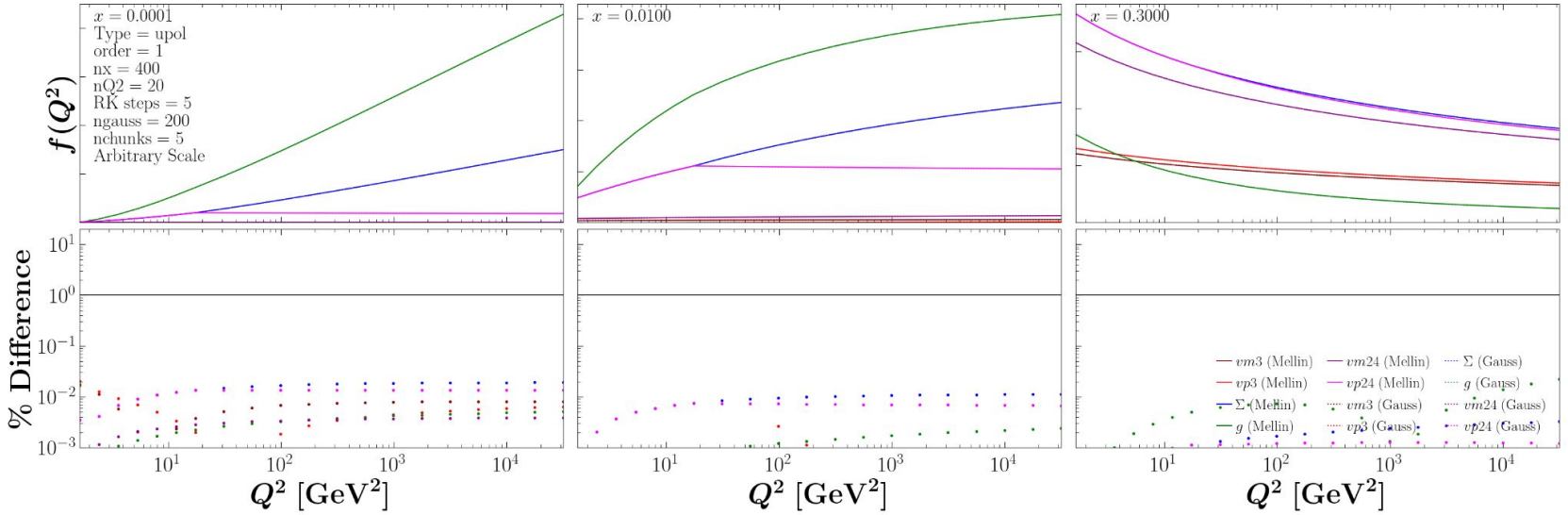


No Matrix



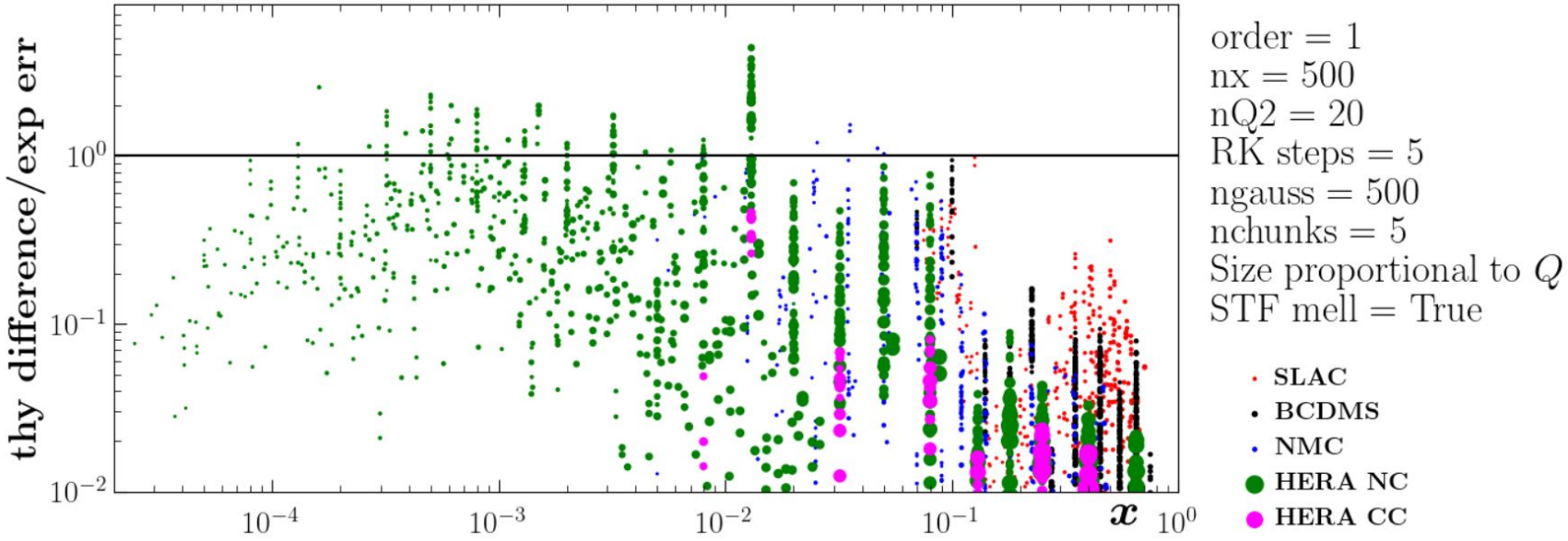
← Benchmark result - can't do better than this

Accuracy of Full Evolution



Benchmark against Mellin result, which we know can reproduce fixed coupling result to high accuracy

Comparing Uncertainty of Theory and Experiment



Theory uncertainty estimated from difference between Mellin and *x*-space results

Evolution in the time it takes to do matrix multiplication

$$f^i(t + N\Delta t) \approx E^{ij}(t + N\Delta t) f^j(t)$$

E can be precomputed. The evolution to any energy scale of *any* initial condition takes as long as one matrix multiplication

- APFEL (2014) were the first to do this. They compute matrix evolution on an interpolation basis (of Lagrange polynomials), in contrast with our boxcar functions (essentially just grid points)
- Adam Freese does this with an ‘inter-pixel’ interpolation basis
- They have the advantage of being able to use adaptive quadrature, but their matrix construction is slower.

Other applications: GPDs

This framework works well for GPDs. Adam Freese or Marco Zaccheddu will probably talk about this in a couple of weeks

Not an application: Non-Linear Evolution

We've relied heavily on the property of linearity to derive everything here. As such, I don't know if we can do the same thing for small-*x*/ saturation physics, whose evolution is ultimately non-linear