Gaussian Processes for Higgs inclusive

Jesús Urtasun Elizari

Milan, April 2020

1 Introduction

The main goal of this project is to develop an approximate expression to the Higgs inclusive cross section in gluon fusion up to N3LO. For this purpose we develop a Gaussian Process (GP) regressor, a machine learning model that performs interpolation assuming Gaussian distributed input data, and optimizes the so-called covariance functions, or also *kernels*, by training on the asymptotic expressions.

After testing a simple version of this gaussian regression, also known as *Kringing*, in the $p_\perp$ distribution at leading order in $p_\perp$, we aim to improve the performance given the deeper knowledge and various physical constrains that are available in the inclusive case. The roadmap that we follow is shown below.

1. Generate the NLO, NNLO, N3LO coefficient functions using the numerical implementation done in ggHiggs code by Marco Bonvini.
2. Implement the analytic expressions of the asymptotics at the different perturbative orders from 1303.3590. They will be used as training data as we will explain later in section 4.
3. Build the Gaussian Process regressor.
4. Hyperoptimize the Gaussian Process regressor.

The last step is indeed were we aim to obtain the best performance. By performing hyper optimization, or *hyperopt* on the kernels and internal parameters of the gaussian regressor, we scan the whole parameter space to find the best model for making predictions in our particular process. That way the performance will be versatile enough to be extended not only to different perturbative orders in our inclusive case, but also for making predictions in other processes as $p_\perp$ distribution in a straightforward way.

2 Higgs inclusive in gluon fusion

Our starting point will be to consider the general expression of the inclusive cross section written in terms of the PDFs and the partonic cross section:

$$\sigma(\tau, m_H^2) = \tau \sum_{i,j} \int_\tau^1 \frac{dz}{z}\, \mathcal{L}_{ij}\left(\frac{\tau}{z}, \mu_R^2\right) \frac{1}{z} \hat{\sigma}_{ij}\left(z, m_H^2, \alpha_s(\mu_R^2), \frac{m_H^2}{\mu_F^2}, \frac{m_H^2}{\mu_R^2}\right), \qquad \tau = \frac{m_H^2}{s} \tag{1}$$

being $\mathcal{L}_{ij}(z, \mu^2)$ the parton luminosity tensors

$$\mathcal{L}_{ij}(z, \mu^2) = \int_z^1 \frac{dx}{x} f_i\left(\frac{z}{x}, \mu^2\right) f_j(x, \mu^2) \tag{2}$$

Write the cross section in terms of the so called coefficient functions C_{ij}

$$\hat{\sigma}_{ij}\left(z, m_H^2, \alpha_s(\mu_R^2), \frac{m_H^2}{\mu_F^2}, \frac{m_H^2}{\mu_R^2}\right) = z\ \sigma_0\big(m_H^2, \alpha_s(\mu_R^2)\big)\ C_{ij}\left(z, \alpha_s, \frac{m_H^2}{\mu_F^2}, \frac{m_H^2}{\mu_R^2}\right) \tag{3}$$

For our further purposes we will work in Mellin space, therefore the function to approximate can be written as follows showing explicitly the powers of $\ln N$, whose behavior is already well known from both small N and large N (soft) resummation.

$$C_{\text{res}}(N, \alpha_s) = g_0(\alpha_s)\left[\frac{1}{\alpha_s} g_1(\alpha_s \ln N) + g_2(\alpha_s \ln N) + \alpha_s g_3(\alpha_s \ln N) + ...\right] \tag{4}$$

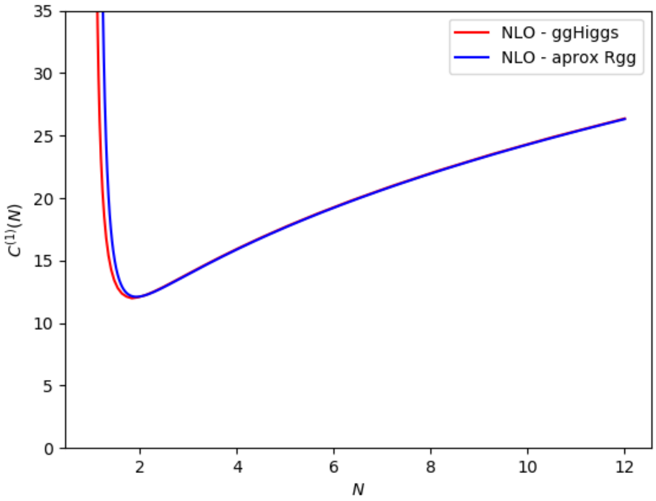


Figure 1: Comparison between our mathematica implementation of $C^{(1)}(N)$, using the approximate form of R_{gg} in the point-like limit, and ggHiggs, which numerically implements the exact expression. As expected, the deviation starts at small N, where the R_{gg} function dominates the behavior.

with

$$g_0(\alpha_s) = 1 + \alpha_s g_{0,1} + \alpha_s^2 g_{0,2} + \mathcal{O}(\alpha_s^3) \tag{5}$$

As explained in section 2 of 1303.3590, we can use an approximate expression as if we knew the N space resummed coefficient function. Then we could extract from eq. (4) the $\mathcal{O}(\alpha_s)$ coefficient $C^{(1)}$ as follows

$$C^{(1)}(z) = 4A_g(z)\mathcal{D}_1(z) + d\delta(1-z) - 2A_g(z)\frac{\ln z}{1-z} + R_{gg}(z), \tag{6}$$

$$\mathcal{D}_k(x) \equiv \left(\frac{\ln^k(1-x)}{1-x}\right)_+, \tag{7}$$

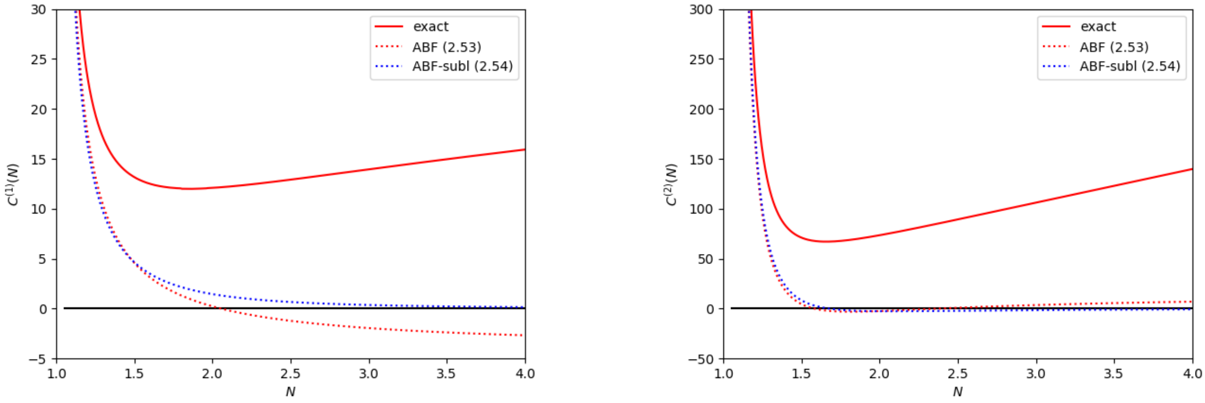
$$A_g(z) = \frac{C_A}{\pi}\frac{1-2z+3z^2-2z^3+z^4}{z}. \tag{8}$$

The behavior at small N is dominated by the R_{gg} function, which has a complicated structure, and therefore the way we obtain $C^{(n)}(N)$ for higher perturbative orders in N space is by using the numerical implementation performed with ggHiggs code by Marco Bonvini.

As a warm up exercise we Mellin transform the coefficient function shown above and compare it with the implementation in Bonvini's code. As expected, as long as we increase N and the behavior stops being dominated by R_{gg}, both curves show excellent numerical agreement.

3 Asymptotics

As mentioned before, the asymptotics are computed by following 1303.3590. Further discussions concerning the N3LO computation can be found in the updated version 1404.3204 We will devote a section below to explain the way the Python implementation was carried out, trying to make it look as clear and user-friendly as possible, since the amount of long algebraic expressions could make non-trivial to follow the calculation for someone not experienced enough with notation.



(a) NLO

(b) NNLO

Figure 2: Different asymptotic expressions at small N together with the exact computation of the Higgs cross section at NLO (left) and NNLO (right), reproducing figure 2 of 1303.3590 with our Python implementation of the asymptotics.

.

3.1 Small N

The leading small N singularities have been determined to all orders in α_s, for both $m_t \to \infty$ and finite m_t. Here we just show the result computed at 1303.3590. By combining the expansion terms of the anomalous dimension γ^+

$$\gamma^{(0)} = \frac{e_{0,-1}}{N-1} + e_{00} + \mathcal{O}(N-1) \tag{9}$$

$$\gamma^{(1)} = \frac{e_{1,-2}}{(N-1)^2} + \frac{e_{1,-1}}{N-1} + \mathcal{O}(1) \tag{10}$$

$$\gamma^{(2)} = \frac{e_{2,-3}}{(N-1)^2} + \frac{e_{2,-2}}{N-1} + \mathcal{O}(N-1)^{-1} \tag{11}$$

where the coefficients $e_{i,j}$ are constants depending on the C_A, C_F color factors and the number of flavors n_f, all listed in the appendix. The small N approximation to the coefficient function is obtained as

$$\begin{aligned}
C_{\rm ABF}(N,\alpha_s) &= \sum_{n=1}^{\infty} \alpha_s^n C^{(n)}_{\rm ABF}(N) && (12)\\
&= \sum_{i_1,i_2\geq 0} c_{i_1,i_2}\left[\gamma^{+i_1}\right]\left[\gamma^{+i_2}\right] - 1 && (13)\\
&= \alpha_s 2c_{1,0}\gamma^{(0)} && (14)\\
&\quad + \alpha_s^2\left[(2c_{2,0}+c_{1,1})\gamma^{(0)^2} - 2c_{2,0}\beta_0\gamma^{(0)} + 2c_{1,0}\gamma^{(1)}\right]\\
&\quad + \alpha_s^3\Bigg[(c_{3,0}+c_{2,1})2\gamma^{(0)^3} - 3(c_{3,0}+c_{2,1})2\beta_0\gamma^{(0)^2} + 4c_{3,0}\beta_0^2\gamma^{(0)} && (15)\\
&\qquad\qquad + (2c_{2,0}+c_{1,1})2\gamma^{(0)}\gamma^{(1)} - 4c_{2,0}\beta_0\gamma^{(1)} + 2c_{1,0}\gamma^{(2)}\Bigg]\\
&\quad + \mathcal{O}(\alpha_s^4)
\end{aligned}$$

We can also use an improved, or *subleading* version, built by subtracting the expression above in such a way that it keeps the small N behavior but that ensures vanishing behavior also at $N \to \infty$.

$$C^{(n)}_{\text{ABF-subl}}(N) = C^{(n)}_{\text{ABF}}(N) - 2C^{(n)}_{\text{ABF}}(N+1) + C^{(n)}_{\text{ABF}}(N+2) \tag{16}$$

The subtraction only introduces subleading $N = 0$ and $N = -1$ singularities. Different possible choices are perfectly allowed on how to perform the subtraction.

3.2 Large N

In the large N (soft) approximation we find powers of $\ln N$ which are just the large N approximation of the digamma function $\psi_0(N)$ appearing in fix order computations. This part of the coefficient function is written as combinations of the $\mathcal{D}_k(N)$, $\hat{\mathcal{D}}_k(N)$, $\mathcal{D}_k^{log}(N)$ distributions, which involve the polygamma functions as defined in Appendix (A.6) of 1303.3590. We show here their definition in direct x space

$$\mathcal{D}_k(x) \equiv \left(\frac{\ln^k(1-x)}{1-x}\right)_+ , \tag{17}$$

$$\mathcal{D}_k^{log}(x) \equiv \left(\frac{\ln^k \ln\frac{1}{x}}{\ln\frac{1}{x}}\right)_+ , \tag{18}$$

$$\hat{\mathcal{D}}_k(x) \equiv \mathcal{D}_k(x) + \left[\frac{\ln^k \ln\frac{1-x}{\sqrt{x}}}{1-x} - \frac{\ln^k(1-x)}{1-x}\right], \tag{19}$$

And their Mellin expressions as defined in appendix A of 1303.3590, or equivalently from appendix B of 1212.0480. By combining them we obtain the $S(N)$ combinations that will be used to write down the soft approximations. From (2.36)

$$C_{\text{soft1}}(N) = \bar{g}_0(\alpha_s)\exp\sum_{n=1}^{\infty}\alpha_s^n\sum_{k=0}^{n} b_{n,k}\,\hat{\mathcal{D}}_k(N+1) \tag{20}$$

$$C_{\text{soft2}}(N) = \bar{g}_0(\alpha_s)\exp\sum_{n=1}^{\infty}\alpha_s^n\sum_{k=0}^{n} b_{n,k}\left[2\hat{\mathcal{D}}_k(N) - 3\hat{\mathcal{D}}_k(N+1) + 2\hat{\mathcal{D}}_k(N+2)\right] \tag{21}$$

and (2.39)

$$C_{\text{soft}}(N,\alpha_s) = \bar{g}_0(\alpha_s)\exp\sum_{n=1}^{\infty}\alpha_s^n S_n(N) \tag{22}$$

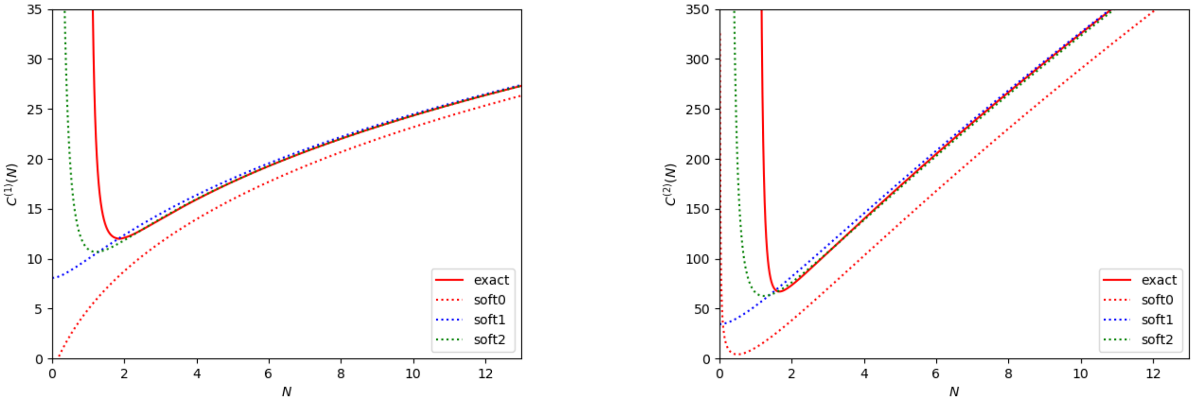
we identify the S_n(N) expressions that lead to the soft approximations at the different perturbative orders. Again, a detailed description of the way all asymptotic expressions are obtained from the paper, leading to reproducing every figure on it and taking full control of the input data for our GP, is explained below. One arrives to build

$$C^{(1)}_{\text{soft}} = S_1(N) + \bar{g}_{0,1} \tag{23}$$

$$C^{(2)}_{\text{soft}} = \frac{1}{2}S_1^2(N) + S_2(N) + \bar{g}_{0,1}S_1(N) + \bar{g}_{0,2} \tag{24}$$

$$C^{(3)}_{\text{soft}} = \frac{1}{6}S_1^3(N) + S_1(N)S_2(N) + S_3(N) + \bar{g}_{0,1}\left(\frac{1}{2}S_1^2(N) + S_2(N)\right) + \bar{g}_{0,2}S_1(N) + \bar{g}_{0,3}. \tag{25}$$

We choose the asymptotic expression labeled as soft$_1$ as the large N asymptotics as training data in that region. At small N, we find no remarkable difference when changing from the C_{ABF} to its subleading version.



(a) NLO

(b) NNLO

Figure 3: Different asymptotic expressions at large N together with the exact computation of the Higgs cross section at NLO (left) and NNLO (right), reproducing figure 1 of 1303.3590 with our Python implementation of the asymptotics.

3.3 N3LO discussion

1. Asymptotics still to be fully implemented for N3LO. Check updated version 1404.3204.

2. Reproduce figures 4 and 5 of 1303.3590 together with figure 4 1805.08785

3. As a next step, try also to interpolate from previous orders given the convergence shown in figure 8 of 1303.3590 rather from the asymptotics.

4 Gaussian Processes for inclusive observables

Now we are ready to start training our machine learning gaussian regressor. This kind of interpolation begins by assuming that the function to guess is sampled from a Gaussian distribution $f(x) \sim N(\mu, K(\theta, x, x'))$, where $K(\theta, x, x')$ is the covariance matrix between all possible points x, x' given a set of parameters θ (such as the length scale of the kernel, periodicity if any, or more specific parameters concerning the definition of the kernels). The way we will apply the interpolation goes as follows

1. Take the asymptotics computed at check_n3lo_data.py by following 1303.3590 as input (training) data.

2. Assume the points x are sampled from a Gaussian distribution $f(x) \sim N(\mu, K(\Theta, x, x'))$.

3. Train Θ by minimizing a $\chi^2(f(x)|\Theta, x)$ to guess the underlying law.

4. Use the Mellin-space exact coefficient functions computed by ggHiggs as true values for testing and validation.

4.1 Implement asymptotics

As said before, we devote a section to provide a detailed description of the way the Python implementation of the asymptotics was carried out, and its comparison with the exact coefficient functions, trying to make it look as accessible and user-friendly as possible towards other possible applications. On the asymptotics folder of the N3PDF repository they are organize as follows.

1. The main file check_n3lo_data.py reproduces the different figures in the paper by first loading the exact curves from ggHiggs and then calling the asymptotic expressions.

2. The contributions_small_n.py file contains the expansions of the anomalous dimension $\gamma^{(i)}$ together with the small N asymptotics labeled as ABF and ABF-subl, following the notation from the paper, up to N3LO.

3. The contributions_large_n.py file holds for the explicit expressions for the $\mathcal{D}_k(N)$, $\mathcal{D}_k^{log}(N)$ and $\hat{\mathcal{D}}_k(N)$ distributions, their combinations $S(N)$ and the large N asymptotics labeled as soft-0, soft-1, soft-2, following the notation from the paper, up to N3LO.

4. File check_data.py computes the comparison shown in figure 1 together with some old checks of the first implementations of the asymptotics.

To begin with we set some ground values for the point at we stop the training asymptotics, defining the interpolation region. They can be changed, together with the number of points used for training, as a next step towards a more fine tuned performance.

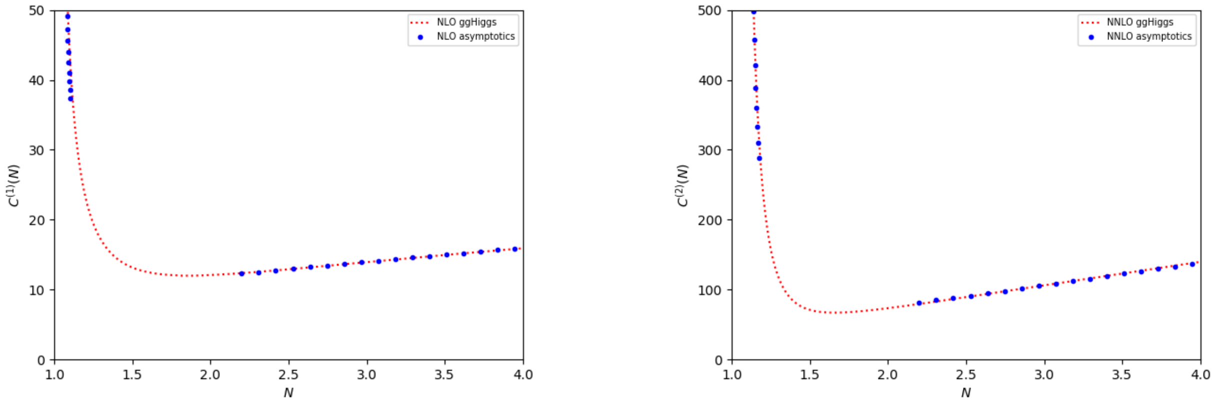


Figure 4: Representation of the asymptotics (training data) and exact inclusive coefficient function (testing and validation) at NLO (left) and NNLO (right) as a function of N.

4.2 Covariance functions

The key object we aim to train for performing an optimal interpolation is the so called *kernel*, or *covariance function*, also known as covariance matrix. The different kernels tried for our process are the following, scikit-learn-kernels

4.2.1 Constant kernel

The Constant kernel is the simplest case. It can be used as part of a product kernel where it scales the magnitude of the other kernel or as part of as part of a sum kernel, where it modifies the mean of the gaussian process. It depends on a parameter C and it is defined as:

$$k(x_i, x_j) = C \; \forall \; x_i x_j \tag{26}$$

4.2.2 RBF

The Radial Basis Function (RFB) kernel is also known as the squared exponential kernel. It is parameterized by a length-scale parameter $l > 0$, which can either be a scalar (isotropic variant of the kernel) or a vector with the same number of dimensions as the inputs (anisotropic variant). For our case lets consider the isotropic variant, where the kernel is given by

$$k(x_i, x_j) = \exp\left(-\frac{1}{2} d\big(x_i/l, x_j/l\big)^2 \right) \tag{27}$$

This kernel is infinitely differentiable, which implies that GPs with this kernel as covariance function have mean square derivatives of all orders, and are thus very smooth.

4.2.3 Matern

The Matern kernel is a stationary kernel and a generalization of the RBF kernel. It has an additional parameter which controls the smoothness of the resulting function. It is parameterized by a length-scale parameter , which can either be a scalar (isotropic variant of the kernel) or a vector with the same number of dimensions as the inputs (anisotropic variant). The kernel is given by

$$k(x_i, x_j) = \sigma^2 \frac{1}{\Gamma(\nu) 2^{\nu-1}} \Big(\gamma \sqrt{2\nu} d\big(x_i/l, x_j/l\big) \Big)^{\nu} K_{\nu} \Big(\gamma \sqrt{2\nu} d\big(x_i/l, x_j/l\big) \Big) \tag{28}$$

when $\nu \to \infty$, the Matern kernel tends to the RBF. Other popular choices are also $\nu = 3/2$ and $\nu = 5/2$, leading to a one-time and two-times differentiable kernel, respectively (add documentation...)

4.2.4 Rational Quadratic

The Rational Quadratic kernel can be seen as a scale mixture, or infinite sum, of RBF kernels with different characteristic length-scales. It is parameterized by a length-scale parameter $l > 0$ and a scale mixture parameter $\alpha > 0$. Only the isotropic variant where l is a scalar is supported at the moment. The kernel is given by:

$$k(x_i, x_j) = \left(1 + \frac{d(x_i, x_j)}{2\alpha l^2}\right)^{-\alpha} \tag{29}$$

4.2.5 Exp-sin

The ExpSineSquared kernel allows modeling periodic functions. It is parameterized by a length-scale parameter $l > 0$ and a periodicity parameter $p > 0$.

$$k(x_i, x_j) = \exp\left(-2\sin(\pi/p \cdot d(x_i, x_j))/l)^2\right) \tag{30}$$

This is an example of the kind of kernel we do not expect to perform in an optimal way to our problem, given the asymmetry of the function to interpolate, and indeed it is shown to be a bad choice in figure (Add figure example ...)

4.2.6 Dot product

The Dot Product kernel can be obtained from linear regression by putting $N(0,1)$ priors on the coefficients of $x_d (d = 1, ..., D)$ and a prior $N(0, \sigma_{0^2})$ on the bias. The Dot Product kernel is invariant to a rotation of the coordinates about the origin, but not translations. It is parameterized by a parameter σ_{0^2}. For $\sigma_{0^2} = 0$, the kernel is called the homogeneous linear kernel, otherwise it is inhomogeneous. The kernel is given by

$$k(x_i, x_j) = \sigma_0^2 + x_i \cdot x_j \tag{31}$$

This kernel is also shown to be far from a good choice, as models involving it lead to extremely high χ^2 after training.

4.3 Hyperoptimization

After running over different kernels we already saw that when applying the gaussian regression for the $p_\perp$ that different choices of the covariance functions lead to very different performance, and we already find some of them performing good interpolation, trained by comparing with data using a χ^2 estimator. As said, over all available kernels and using their default parameters listed in scikit-learn kernels, we see that the ones that perform better given the $1-\chi^2$ score used are the Matern, Rational Quadratic, together with linear combinations and products between them all, as shown in figures 5 and 6.

1. The HyperGP_inclusive_Nspace.py main file performs the hyperopt and the Gaussian Process regression for the Higgs inclusive cross section.
2. The HyperGP_pt_distribution.py file performs the hyperopt and the Gaussian Process regression for the Higgs $p_\perp$ distribution (work in progress).
3. Both files rely on the HyperGP.py class, which we inherit from the Gaussian Process regressor and manually to manually set the desired parameters, such as the kernels, their internal parameters and the way the score is computed, as training parameters to be optimized.
4. The nlo_hyperopt.txt stores all parameter dictionaries, together with their scores achieved during they hyperopt process for the NLO search.
5. The nnlo_hyperopt.txt stores all parameter dictionaries, together with their scores achieved during they hyperopt process for the NNLO search.

The way the hyperoptimization is implemented is by instancing a Grid Search to run non only over the list of mentioned kernels, but scan all parameter space to find the best combination for our model. The different combinations to try are given to the model as a dictionary of parameters to tune. One advantage of the Grid Search implementation is that it allows for the activation of a cross validation mechanism, that performs a validation of every model with some subset of the training set.

The way to perform the search and selection of the best model is by applying cross validation, selecting a fraction of the training data to compare with the exact coefficient function by computing a score $1-\chi^2$. That way, once a model is chosen to train it has already been tested (validated) by computing a prediction for a discrete number of points and comparing it with the validation set. The definition of the training - validation split can be modified as an argument to the file HyperGP_inclusive_Nspace.py, while the specific way the score is defined is set in the class HyperGP.py.

Then the model chosen as the best one is trained performing the usual χ^2 minimization. Results so far are shown in tables 1, 2, together with examples of the performance for different parameter sets in figures 5 and 6.

Kernel	Best parameter set	χ^2
Matern	$l = 0.9, \nu = 0.9$	0.28365
Matern	$l = 1.0, \nu = 0.9$	0.21115
Matern	$l = 1.3, \nu = 0.9$	0.22024

Table 1: Example of the best parameter sets found for interpolation at NLO with with N range (1.075, 2.1), together with the χ^2 achieved after training. As shown, Matern is the best kernel found so far.

Kernel	Best parameter set	χ^2
Matern	$l = 0.8, \nu = 1.0$	0.02413
Matern	$l = 0.9, \nu = 1.0$	0.02069
Sum2	$l = 1.3, \nu = 0.9, \alpha = 0.8$	0.41660

Table 2: Example of the best parameter sets found for interpolation at NNLO with with N range (1.075, 2.1), together with the χ^2 achieved after training. As shown, Matern and Sum2, consisting in the sum of a Matern and a Rational Quadratic kernel, are the best choices found so far.

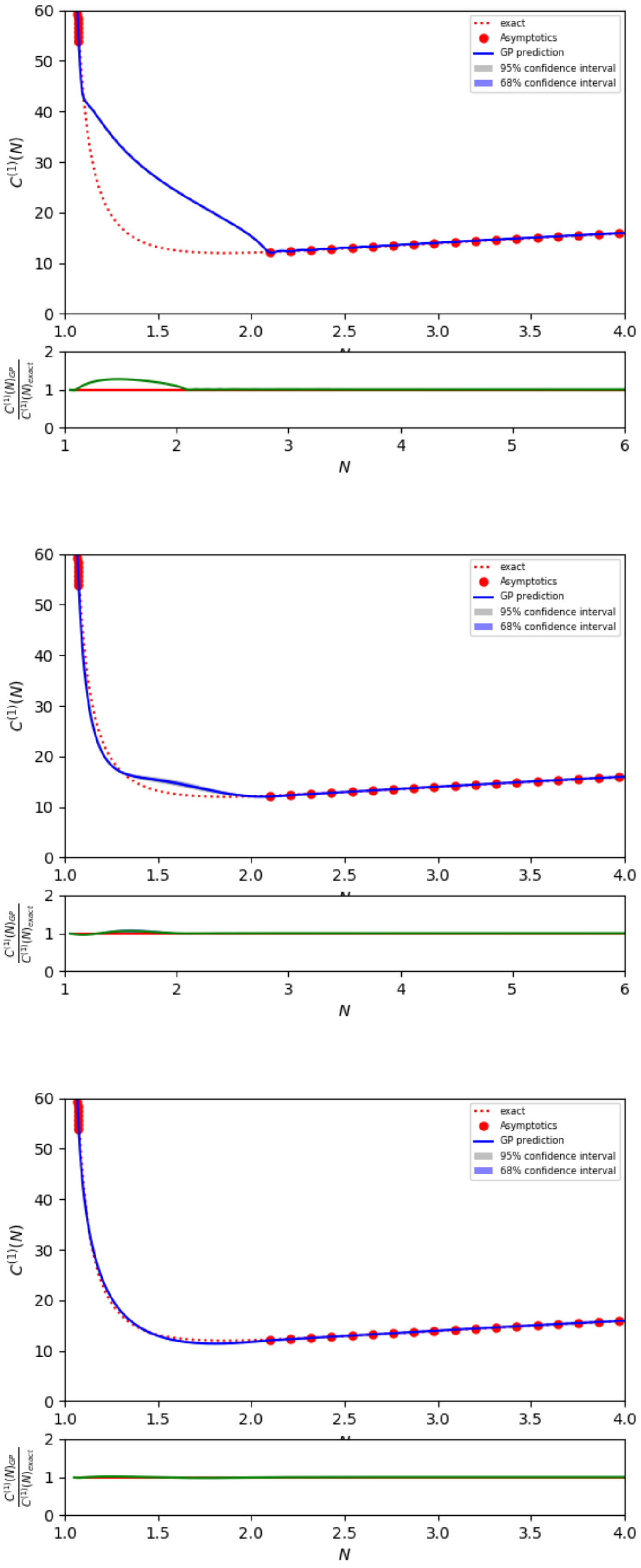


Figure 5: Example of the interpolation performed at NNLO by different parameter sets as a function of N. The example below shows the performance done by the parameter set that obtained the best score so far, with values shown in table 1.

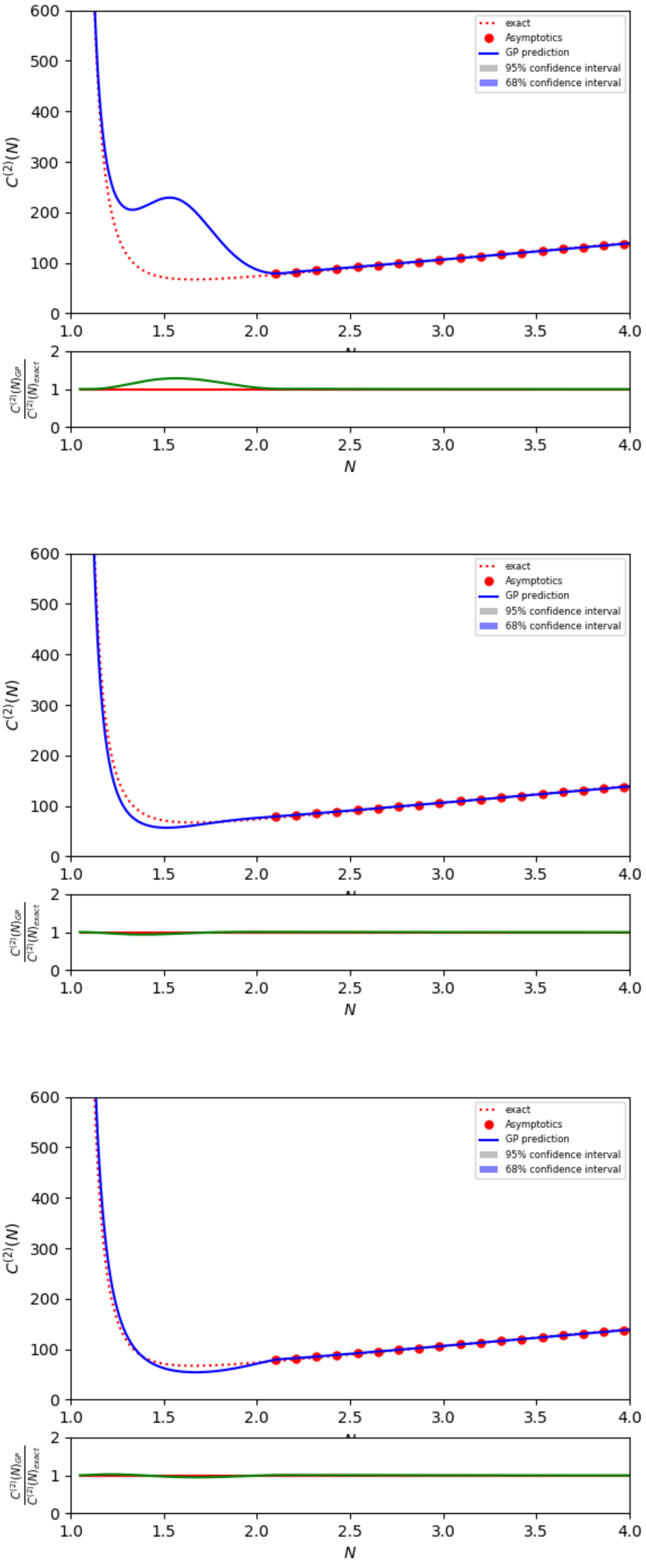


Figure 6: Example of the interpolation performed at NNLO by different parameter sets as a function of N. The example below shows the performance done by the parameter set that obtained the best score so far, with values shown in table 2.