9 Underlying Theory of *Smooth Emulator*

The choice of model emulators, $E(\vec{\theta})$, depends on the prior understanding of the model being emulated, $M(\vec{\theta})$. If one knows that a function is linear, then a linear fit is clearly the best choice. Whereas to reproduce lumpy features, where the lumps have a characteristic length scale, Gaussian process emulators are highly effective. The quality of an emulator can be assessed through the following criteria:

- $E(\vec{\theta}_t) = M(\vec{\theta}_t)$ at the training points, $\vec{\theta}_t$.
- The emulator should reasonably reproduce the model away from the training points. This should hold true for either interpolation or extrapolation.
- The emulator should reasonably represent its uncertainty
- A minimal number of training points should be needed
- The method should easily adjust to larger numbers of parameters, $\theta_i,\ i = 1 \cdots N$
- The emulator should not be affected by unitary transformations of the parameter space
- The emulator should be able to account for noisy models
- Training and running the emulator should not be numerically intensive

Here the goal is to focus on a particular class of functions: functions that are *smooth*. Smoothness is a prior knowledge of the function. It is an expectation that the linear terms of the function are likely to provide more variance than the quadratic contributions, which are in turn likely to be more important than the cubic corrections, and so on.

9.1 Mathematical Form of *Smooth Emulator*

To that end the following form for $E(\vec{\theta})$ is chosen,

$$E(\vec{\theta}) = \sum_{\vec{n},\text{s.t. } K(\vec{n}) \leq K_{\max}} d_{\vec{n}} f_{K(\vec{n})}(|\vec{\theta}|) A_{\vec{n}} \left(\frac{\theta_1}{\Lambda}\right)^{n_1} \left(\frac{\theta_2}{\Lambda}\right)^{n_2} \cdots \left(\frac{\theta_N}{\Lambda}\right)^{n_N}. \tag{9.1}$$

Each term has a rank $K(\vec{n}) = n_1 + n_2 + \cdots n_N$. If f is constant, the rank of that term corresponds to the power of $|\vec{\theta}|/\Lambda$. All terms are included up to a given rank, $K_{\max}$. The coefficients A are stochastically distributed. The coefficients $d_{\vec{n}}$ will ensure that the variance is independent of the direction of $\vec{\theta}$, with the constraint that $d_{K,0,0\cdots} = 1$. The function $f_K(|\vec{\theta}|)$ provides the freedom to alter how the behavior depends on the distance from the origin, $|\vec{\theta}|$, and on the rank, K. Given that the variance of $A_{\vec{n}}$ can be changed, $f_{K=0}(|\vec{\theta}| = 0)$ is also set to unity for all K without loss of generality. For each combination $\vec{n}$, the prior probability for any the A coefficients is given by

$$\begin{aligned} p(A_{\vec{n}}) &= \frac{1}{\sqrt{2\pi\sigma^2_{K(\vec{n})}}} e^{-A^2_{\vec{n}}/2\sigma^2_{K(\vec{n})}}, \\ \langle A^2_{\vec{n}} \rangle &= \sigma^2_{K(\vec{n})}. \end{aligned} \tag{9.2}$$

The variance, σ_K^2, is allowed to vary as a function of K.

The parameter Λ will be referred to as the *smoothness parameter*. Here, we assume that all parameters have a similar range, of order unity, e.g. $-1 < \theta_i < 1$. Thus, the relative importance of each term Eq. (9.1) falls with increasing rank, K, as $(1/\Lambda)^K$. For now, the smoothness parameter is fixed by prior knowledge, i.e. one chooses higher values of Λ if one believes the function to be close to linear.

First, we consider the variance of the emulator at a given point, $\vec{\theta}$. Requiring that the variance is independent of the direction of $\vec{\theta}$ will fix $d_{\vec{n}}$. For example, transforming θ_1 and θ_2 to parameters $(\theta_1 \pm \theta_2)/\sqrt{2}$ should not affect the accuracy or uncertainty of the emulator.

At $|\vec{\theta}| = 0$ the only term in Eq. (9.1) that contributes to the variance is the one $K = 0$ term. Averaging over the A coefficients, which can be either positive or negative with equal probability,

$$\langle E(\vec{\theta}) \rangle = 0, \tag{9.3}$$

where the averaging refers to an average over the A coefficients. At the origin, $|\vec{\theta}| = 0$, the variance of E is

$$\langle E(\theta_1 = \theta_2 = \cdots \theta_N = 0)^2 \rangle = d^2_{n_i=0} \sigma^2_{K=0} f^2_{K=0}(\vec{\theta} = 0). \tag{9.4}$$

Choosing $f_{K=0}(0) = 1$ and $d_{n_i=0} = 1$, the variance of E is indeed σ_0^2. The variance at some point $\vec{\theta} \neq 0$ is

$$\langle E^2(\vec{\theta}) \rangle = \sum_{\vec{n}} f_K^2(|\vec{\theta}|) \sigma^2_{K(\vec{n})} d^2_{\vec{n}} \left(\frac{\theta_1^{2n_1}}{\Lambda^2} \right) \left(\frac{\theta_2^{2n_2}}{\Lambda^2} \right) \cdots \left(\frac{\theta_N^{2n_N}}{\Lambda^2} \right). \tag{9.5}$$

If $\langle E^2 \rangle$ is to be independent of the direction of $\vec{\theta}$, the sum above must be a function of $|\vec{\theta}|^2$ only. This requires the net contribution from each rank, K to be proportional to $|\vec{\theta}|^{2K}$ multiplied by some function of K. Using the fact that

$$(\vec{\theta}_a \cdot \vec{\theta}_b)^K = \sum_{\vec{n}, s.t. \sum_i n_i = K} \frac{K!}{n_1! \cdots n_N!} (\theta_{a1}\theta_{b1})^{n_1} \cdots (\theta_{aN}\theta_{bN})^{n_N}, \tag{9.6}$$

one can see that if the sum is to depend only on the norm of $\vec{\theta}$,

$$d^2_{\vec{n}} = \frac{K(\vec{n})!}{n_1! n_2! \cdots n_N!}. \tag{9.7}$$

The factor of $K!$ in the numerator was chosen to satisfy the condition that $d_{K,0,0,0} = 1$.

One can now calculate the correlation between the emulator at two different points, averaged over all possible values of A,

$$\langle E(\vec{\theta}_a) E(\vec{\theta}_b) \rangle = \sum_{K=0}^{K_{\max}} \sigma_K^2 f_K^2(|\vec{\theta}|) \left(\frac{\vec{\theta}_a \cdot \vec{\theta}_b}{\Lambda^2} \right)^K. \tag{9.8}$$

Requiring $f(|\theta|=0)=1$ gives

$$\langle E^2(\vec{\theta}=0))\rangle=\sigma_0^2, \tag{9.9}$$

and for $\vec{\theta}_a=\vec{\theta}_b$,

$$\langle E^2(\vec{\theta}=0))\rangle \;\; =\sum_{K=0}^{K_{\rm max}}\sigma_K^2 f_K^2(|\vec{\theta}|)\left(\frac{|\vec{\theta}|^2}{\Lambda^2}\right)^K. \tag{9.10}$$

To this point, the form is completely general once one requires that the variance above is independent of the direction of $\vec{\theta}$. I.e. the function $f_K(\vec{\theta})$ could be any function satisfying the constraint, $f_K(0)=1$, and σ_K^2 could have any function of K. Below, we illustrate how different choices for f or for σ_K affect the emulator by comparing several variations. First, we define the default form.

9.2 Alternate Forms

As stated above, once the form is to provide variances that are independent of the direction of $\vec{\theta}$, the general form going forward is

$$\begin{aligned}E(\vec{\theta}) &= \sum_{\vec{n},{\rm s.t.}\; K(\vec{n})<K_{\rm max}} f_K(|\vec{\theta}|)\left(\frac{K(\vec{n})!}{n_1!\cdots n_N!}\right)^{1/2}A_{\vec{n}}\left(\frac{\theta_1}{\Lambda}\right)^{n_1}\left(\frac{\theta_2}{\Lambda}\right)^{n_2}\cdots\left(\frac{\theta_N}{\Lambda}\right)^{n_N},\\ P(\vec{A}_n) &= \frac{1}{(2\pi\sigma_K^2)^{1/2}}e^{-|A_{\vec{n}}|^2/2\sigma_K^2}.\end{aligned} \tag{9.11}$$

Variations from the general form involve adjusting either the K-dependence of the $|\vec{\theta}|$-dependence of $f_K(|\vec{\theta}|)$, or adjusting the K-dependence of σ_K.

Default Form
Here, we assume $f_K(|\vec{\theta}|)$ is independent of $|\vec{\theta}|$, and that σ_K is independent of K. Further, the $K-$dependence of f^2 is assumed to be $1/K!$. With this choice

$$\begin{aligned}E(\vec{\theta}) &= \sum_{\vec{n},K(\vec{n})\leq K_{\rm max}}\frac{1}{\Lambda^K}\frac{A_{\vec{n}}}{\sqrt{n_1!n_2!\cdots n_N!}}\theta_1^{n_1}\cdots\theta_N^{n_N},\\ P(A_{\vec{n}}) &\sim e^{-A_{\vec{n}}^2/2\sigma^2}.\end{aligned} \tag{9.12}$$

With this form the variance increases with $|\vec{\theta}|$,

$$\langle E^2(\vec{\theta})\rangle \;\; =\sigma^2 e^{|\vec{\theta}|^2/\sigma^2}. \tag{9.13}$$

If the function is trained in a region where the function is linear, the emulator's extrapolation outside the region will continue to be follow the linear behavior, albeit with variation from the higher order coefficients.

The choice of $f_K^2=1/K!$ ensures that the sum defining $E(\vec{\theta})$ converges as a function of K as long as $K_{\rm max}$ is rather large compared to $|\vec{\theta}|/\Lambda$.

Variant A: Letting σ_K have a K dependence
One reasonable alteration to the default choice might be to allow the $K = 0$ term to take any value, i.e. $\sigma_{K=0} = \infty$, while setting all the other σ_K terms equal to one another. This would make sense if our prior expectation of smoothness meant that we expect the $K = 2$ terms to be less important than the $K = 1$ terms, by some factor $|\vec{\theta}|/\Lambda$, but that the variation of the $K = 1$ term is unrelated to the size of the $K = 0$ term. This would make the emulator independent of redefinition of the emulated function by adding a constant. This may well be a reasonable choice for many circumstances.

Variant B: Suppressing correlations for large $\Delta\vec{\theta}$
This form for f causes correlations to fall for points far removed from one another.

$$f_K(|\vec{\theta}|) = \frac{1}{\sqrt{K!}} \left\{ \sum_{K=0}^{K_{\rm max}} \frac{1}{\sqrt{K!}} \left(\frac{|\vec{\theta}|^2}{2\Lambda^2} \right)^K \right\}^{-1/2} .$$

In the limit that $K_{\rm max} \rightarrow \infty$ the form is a simple exponential,

$$f_K(|\vec{\theta}|)\Big|_{K_{\rm max}\rightarrow\infty} = \frac{1}{\sqrt{K!}} e^{-|\vec{\theta}|^2/2\Lambda^2}. \tag{9.14}$$

With this form the same-point correlations remain constant over all $\vec{\theta}$,

$$\langle E(\vec{\theta}) E(\vec{\theta}) \rangle = \sigma^2, \tag{9.15}$$

while the correlation between separate positions fall with increasing separation. This is especially transparent for the $K_{\rm max} \rightarrow \infty$ limit,

$$\langle E(\vec{\theta}_a) E(\vec{\theta}_b) \rangle_{K_{\rm max}\rightarrow\infty} = \sigma^2 e^{-|\vec{\theta}_a - \vec{\theta}_b|^2/2\Lambda^2}.$$

In this limit one can also see that

$$\langle [E(\vec{\theta}) - E(\vec{\theta}')]^2 \rangle_{K_{\rm max}\rightarrow\infty} = 2\sigma^2 \left(1 - e^{|\vec{\theta}-\vec{\theta}'|^2/2\Lambda^2} \right). \tag{9.16}$$

If one trains such an emulator in one region, then extrapolates to a region separated by $|\vec{\theta} - \vec{\theta}'| >> \Lambda$, the average predictions will return to zero. Thus, if the behavior would appear linear in some region the emulator's distribution of predictions far away (extrapolations) would center at zero. This behavior is similar to a Gaussian-process emulator.

Variant C: Eliminating the $1/K!$ weight
Clearly, eliminating the $1/K!$ weights in f_K would more emphasize the contributions from larger K. But, for $|\vec{\theta}| > \Lambda$ the contribution to the variance would increase as K increases and the sum would not converge if $K_{\rm max}$ were allowed to approach infinity. An example of a function that expands without factorial suppression is $1/(1 - x) = 1 + x + x^2 + x^3 \cdots$, which diverges as $x \rightarrow 1$. If such behavior is not expected, then this choice would be unreasonable.

9.3 Tuning the Emulator via MCMC

Here, we illustrate how an emulator of the form above can be constrained given training points. We consider $N_{\rm train}$ full model runs at positions $\vec{\theta}_a,\ a = 1, N_{\rm train}$, with values Y_a. The functional form has a large number of coefficients, $A_{\vec{n}}$. However, the dependence of A is purely linear. One can denote the coefficients as A_i with $i = 1 \cdots N_{\rm coef}$, where $N_{\rm coef} >> N_{\rm train}$. One can randomly set the coefficients $A_i,\ i = N_{\rm train} + 1 \cdots N_{\rm coeff}$ then determine the first $N_{\rm train}$ coefficients by solving a linear equation. One can then apply a weight based on the values of A consistent with the prior likelihood of A and the constraints. One can then generate a representative set of A, perhaps a dozen samples. Each sample function will go through all the training points, but will vary further from the training points. Averaging over the $N_{\rm sample}$ sets of coefficients can be used to make a prediction for the emulator at some point $\vec{\theta}$, and the variance of those $N_{\rm sample}$ points would represent the uncertainty of the emulator.

Here, we present two different methods for generating samples of points that are consistent with the training constraints and with the prior range of parameters. We do this for the default method, but this can be easily extended to the other model variants listed in the previous section. In addition to varying the coefficients, we will additionally vary the width parameters, σ.

First, we need to describe how the weights are generated. The probability of a set of coefficients is initially

$$P(\vec{A}) = \prod_c \frac{1}{\sqrt{2\pi\sigma^2}} e^{-A_c^2/2\sigma^2}. \tag{9.17}$$

If we also vary σ, we can choose a prior probability for σ. E.g.

$$Q(\sigma) = \frac{1}{\pi} \frac{\Gamma/2}{\sigma^2 + (\Gamma/2)^2}. \tag{9.18}$$

Here, the half width of the distribution should be set to some large value to encompass the degree to which the emulated function might vary. The Lorentzian form accommodates a large uncertainty. The result should not depend strongly on Γ. One can also choose a flat distribution. The joint probability is $Q(\sigma) \prod_c P(A_c)$.

The constrained probability is

$$\begin{aligned} dP &= d\sigma Q(\sigma) \prod_{c=1}^{N_{\rm coef}} [dA_c\, P(A_c)] \prod_{m=1}^{N_{\rm train}} \delta(E(A, \sigma, \vec{\theta}_m) - F(\vec{\theta}_m)) \qquad (9.19) \\ &= d\sigma Q(\sigma) \prod_{c \le N_{\rm train}} P(A_c) \frac{1}{|J|} \prod_{c=N_{\rm train}}^{N_{\rm coef}} [dA_c\, P(A_c)]. \end{aligned}$$

Here, $|J|$ is the Jacobian, i.e. it is the determinant of the $N_{\rm train} \times N_{\rm train}$ matrix

$$J_{mc} = \frac{\partial E(\vec{\theta}_m)}{\partial A_c}, \tag{9.20}$$

For the default form in the previous section,

$$J_{mc} \ = \frac{1}{\sqrt{n_{c1}! \cdots n_{cN}!}} (\theta_{m1})^{n_{c1}} \cdots (\theta_{mN})^{n_{cN}}. \tag{9.21}$$

Because the Jacobian depends only on the position of the training points, it can be treated as a constant.

One can now sample A and σ according to the weights above. Here, we list two methods.

a) **Keep and Reject Method**
One can generate the coefficients, $A_i,\ i > N_{\rm train}$, according to the Gaussian distribution with width σ, after generating σ according to the Lorentzian. The residual weight is then

$$w = \prod_{c=1}^{N_{\rm coef}} P(A_c). \tag{9.22}$$

For each attempt, one could keep or reject the attempt with probability w. This generates perfectly independent samples, but with the caveat that in a high dimensional space the rejection level becomes untenably high.

b) **Metropolis Exploration of A and σ**
Beginning with any configuration A and width σ, one can take a random step δA and $\delta\sigma$. In the absence of any weight this would consider all A or σ with values from $-\infty$ to ∞. The residual weight would then be

$$w = Q(\sigma) \prod_c P(A_c). \tag{9.23}$$

One then keeps the step if the new weight exceeds the previous one, or if the ratio of the new weight to the old weight is greater than a random number between zero and unity. After some number of steps, $N_{\rm steps}$, one saves the configuration as a representative sample for the emulator. The disadvantage of this approach is that many steps may be needed to ensure that the samples are independent, and thus faithfully represent the variation in the emulator.

9.4 Exact Solution for the Most Probable Coefficients and the Uncertainty

Here, we demonstrate how one can solve for the set of most probable coefficients, $A_{\vec{n}}$, given the the training values, $y_t(\vec{\theta}_t)$, where $\vec{\theta}_t$ are the points at which the emulator was trained. Again, we consider $N_{\rm train}$ to be the number of training points and $N_{\rm coef}$ to be the number of coefficients, which is much larger than $N_{\rm train}$. Given the coefficients, $\vec{A}_i$ for $i > N_{\rm train}$, the first coefficients, $i = 1 \cdots N_{\rm train}$, are found by solving the linear equations for $A_a, a \leq N_{\rm train}$. For shorthand, we define the function $T_a(\vec{\theta})$,

$$E(\vec{\theta}) \;\; = \;\; \sum_{i=1}^{N_{\rm coeff}} A_i \mathcal{T}_i(\vec{\theta}), \tag{9.24}$$

The functions $\mathcal{T}_i(\vec{\theta})$ reference the factors in Eq. (9.12) if the default form is used. Evaluated at the $N_{\rm train}$ training points, we define a $N_{\rm coef} \times N_{\rm train}$ matrix,

$$T_{ai} \equiv \mathcal{T}_i(\vec{\theta}_a), \tag{9.25}$$

where $\vec{\theta}_a$ labels the training points, $a \leq 0 < N_{\rm train}$, and $N_{\rm train} << N_{\rm coef}$. Further, moving forward it is convenient to define the $N_{\rm train} \times N_{\rm train}$ sub-matrix,

$$\tilde{T}_{ab} \;\; = \;\; T_{ab}, \;\; a, b < N_{\rm train}. \tag{9.26}$$

Given the location of the training points, one knows T_{ta}. One can then solve the linear system of equations to find $A_{a\leq N_{\rm train}}$ if one knows $y_{t\leq N_{\rm train}}$. To fit the training points,

$$A_a \;=\; \sum_t \tilde{T}^{-1}_{at}\left(y_t - \sum_{i>N_{\rm train}} T_{ti}A_i\right), \tag{9.27}$$

for $a = 1\cdots N_{\rm train}$. As mentioned above, the matrix $\tilde{T}_{ab}$ is a $N_{\rm train}\times N_{\rm train}$ square matrix. For the purposes of brevity going forward, we define the vector $\alpha_a, a \leq N_{\rm train}$ and $\beta_{a,i},\; i > N_{\rm train}$, so that After making the following definitions,

$$\begin{aligned} \alpha_a &\equiv \sum_{b\leq N_{\rm train}} \tilde{T}^{-1}_{ab}y_b, \\ \beta_{ai} &\equiv \sum_{t'\leq N_{\rm train}} \tilde{T}^{-1}_{at'}T_{t'i}, \end{aligned} \tag{9.28}$$

the expression needed to solve for $A_{a\leq N_{\rm train}}$ can be expressed as

$$A_a \;=\; \alpha_a - \sum_{i>N_{\rm train}} \beta_{ai}A_i. \tag{9.29}$$

All the dependence on the full model is contained within α_a, whereas β_{ai} depends only on the positions of the training points.

9.4.1 Finding the Optimum Coefficients

The goal is to find the coefficients, $A_{i>N_{\rm train}}$, that maximizes the probability given that the first $N_{\rm train}$ coefficients are determined by Eq. (9.29),

$$\begin{aligned} P(\vec{A}) &= \frac{1}{(2\pi\sigma^2)^{N_{\rm coef}/2}}\int \prod_{t\leq N_{\rm coef}} \delta[y_m(\vec{\theta}_t) - E(\vec{\theta}_t)]\prod_i dA_i\; \exp\left\{-\frac{1}{2\sigma^2}A_i^2\right\} \\ &= \frac{1}{(2\pi\sigma^2)^{N_{\rm coef}/2}}\int e^{-\sum_j A_j^2/2\sigma^2}\frac{1}{|J|}\prod_{i>N_{\rm train}} dA_i. \end{aligned} \tag{9.30}$$

Here, $|J|$ is the Jacobian, i.e. it is the determinant of the $N_{\rm train}\times N_{\rm train}$ matrix

$$J_{mc} = \frac{\partial E(\vec{\theta}_m)}{\partial A_c}, \tag{9.31}$$

For the default form in the previous section,

$$J_{mc} \;=\; \tfrac{1}{\sqrt{n_{c1}!\cdots n_{cN}!}}(\theta_{m1})^{n_{c1}}\cdots(\theta_{mN})^{n_{cN}}. \tag{9.32}$$

Because the Jacobian depends only on the position of the training points, it can be treated as a constant. As was discussed earlier, the Jacobian depends on the positions of the training points, $\vec{\theta}_t$, but does not depend on $\vec{A}$. In this case we will also fix σ, so we need merely minimize the argument of the exponential. The argument of the exponential, which is to be minimized, then becomes

$$\begin{aligned} \mathrm{MIN} &= \sum_{i>N_{\rm train}} A_i^2 + \sum_{a\leq N_{\rm train}} \left[\sum_t \tilde{T}^{-1}_{at} \left(y_t - \sum_{i>N_{\rm train}} T_{ti} A_i \right) \right]^2 \qquad (9.33) \\ &= \sum_{i>N_{\rm train}} A_i^2 + \sum_{a\leq N_{\rm train}} \left[\alpha_a - \sum_{i>N_{\rm train}} \beta_{ai} A_i \right]^2 \end{aligned}$$

One next needs to minimize this for each coefficient, $A_{i>N_{\rm train}}$. This gives

$$0 = A_i - \sum_{a\leq N_{\rm train}} \left[\alpha_a - \sum_{j>N_{\rm train}} \beta_{aj} A_j \right] \beta_{ai}. \qquad (9.34)$$

$$\sum_{a\leq N_{\rm train}} \alpha_a \beta_{ai} = \sum_{j>N_{\rm train}} \left(\delta_{ij} + \sum_{a\leq N_{\rm train}} \beta_{ai}\beta_{aj} \right) A_j \qquad (9.35)$$

This would be a straight forward solution, but if there were large numbers of parameters, one might have tens of thousands of coefficients. To avoid solving such a large number of simultaneous equations, one can realize that if one thinks of $A_{i>N_{\rm train}}$ as a high dimensional vector, the only other vectors in that space, i.e. those with index $i > N_{\rm train}$, that can represent the solution are the $N_{\rm train}$ vectors β_{ai}. Thus the solution should have the form

$$A_i = \sum_a \gamma_a \beta_{ai}. \qquad (9.36)$$

One can then solve for γ_a, which will require solving $N_{\rm train}$ simultaneous equations. For the sake of brevity, gall the indices labels, a, b, c or t will vary from $1 - N_{\rm train}$ and those labeled i, j will vary from $N_{\rm train} + 1 \cdots N_{\rm coef}$. One must minimize

$$\mathrm{MIN} = \sum_i \left[\sum_a \gamma_a \beta_{ai} \right]^2 + \sum_a \left[\left(\alpha_a - \sum_i \beta_{ai} \sum_b \gamma_b \beta_{bi} \right) \right]^2. \qquad (9.37)$$

This then gives

$$0 = \sum_b \sum_i \beta_{ai}\beta_{bi}\gamma_b + \sum_{ij} \sum_{b,b'} \beta_{ai}\beta_{bi}\beta_{aj}\beta_{b'j}\gamma_{b'} - \sum_i \beta_{ai} \sum_b \beta_{bi}\alpha_b. \qquad (9.38)$$

Defining

$$B_{ab} \equiv \sum_i \beta_{ai}\beta_{bi}, \qquad (9.39)$$

finding the $N_{\rm train}$ values of γ_b involves solving the set of $N_{\rm train}$ linear equations. Having $a, a', b \leq N_{\rm train}$,

$$\begin{aligned} B_{ab}\gamma_b + \sum_{a'b} B_{aa'}B_{a'b}\gamma_b &= \sum_b B_{ab}\alpha_b, \qquad (9.40) \\ (\delta_{ab} + B_{ab})\gamma_b &= \alpha_a. \end{aligned}$$

Thus, once one has run the full model $N_{\rm train}$ times at the training points, one can find y_t and T_{ai}. Following the prescription above, one then finds α_a and β_{ai}, which then gives B_{ab}. One then solves for γ_b, which when inserted into Eq. (9.36) gives the coefficients A_i for $i > N_{\rm train}$. To find the coefficients, A_a for $a \leq N_{\rm train}$, one then applies Eq. (9.29).

Note that this algorithm never involves a double sum with both indices over all $N_{\rm coef}$. Nor does it involve any matrix manipulations, or linear equation solving involving matrices of $N_{\rm coef} \times N_{\rm coef}$. Assuming that the number of training points is no more than a few hundred, this algorithm should be rather quick.

9.4.2 Finding the Uncertainty when Calculating $E(\vec{\theta})$

Next, one needs to express the uncertainty at a point in parameter space $\sigma_E^2(\vec{\theta}) = \vec{\theta}$, $\langle(\delta E(\vec{\theta}))^2\rangle$. This is given by

$$\sigma_E^2 \;=\; \frac{\partial E}{\partial A_i}\frac{\partial E}{\partial A_j}\langle \delta A_i \delta A_j\rangle. \tag{9.41}$$

Once again, the indices i, j refer to components with $i > N_{\rm train}$. We begin with

$$\begin{aligned} E(\vec{\theta}) &= T_a(\vec{\theta})A_a + T_i(\vec{\theta})A_i, \\ T_a(\vec{\theta}) &= \frac{1}{\Lambda^K}\frac{\theta_1^{n_{1a}}\theta_2^{n_{2a}}\cdots}{\sqrt{n_{1a}!n_{2a}!\cdots}}. \end{aligned} \tag{9.42}$$

If one chooses $\vec{\theta}$ equal to one of the training points, t, then $T_a(\vec{\theta}_t) = T_{ta}$. The differential of $E(\vec{\theta})$ is

$$\delta E(\vec{\theta}) \;=\; \sum_a T_a(\vec{\theta})\delta A_a + \sum_i T_i(\vec{\theta})\delta A_i. \tag{9.43}$$

Substituting for A_a in terms of A_a,

$$\delta E(\vec{\theta}) \;=\; \left[T_i(\vec{\theta}) - T_a(\vec{\theta})\beta_{ai}\right]\delta A_i. \tag{9.44}$$

Next, one needs to calculate $\langle \delta A_i \delta A_j\rangle$. Expanding around the minimum, the exponential of the probability becomes

$$\begin{aligned} P(\delta\vec{A}) &\propto \exp\left\{-\frac{1}{2\sigma^2}\left(\sum_a(\delta A_a)^2 + \sum_i(\delta A_i)^2\right)\right\} \\ &= \exp\left\{-\frac{1}{2\sigma^2}\sum_{ij}\left(\delta_{ij} + \beta_{ai}\beta_{aj}\right)\delta A_i\delta A_j\right\}. \end{aligned} \tag{9.45}$$

Defining

$$D_{ij} \;=\; \delta_{ij} + \beta_i\beta_j, \tag{9.46}$$

the fluctuation of the coefficients is

$$\langle \delta A_i \delta A_j\rangle \;=\; D_{ij}^{-1}. \tag{9.47}$$

Again, if there are many coefficients, inverting the matrix could be numerically costly. But one can realize that the inverse matrix should be of the form

$$D^{-1}_{ij} \;=\; \delta_{ij} + \sum_{ab} \psi_{ab}\beta_{ai}\beta_{bj}, \tag{9.48}$$

and solve for the $N_{\rm train} \times N_{\rm train}$ values of ψ_{ab}. This leads to

$$D^{-1}_{ik}D_{kj} \;=\; \delta_{ij} + \sum_{ab}\beta_{ai}\beta_{bj}(\delta_{ab}+\psi_{ab}) + \sum_{abc}\beta_{ai}\beta_{ak}\beta_{ck}\beta_{bj}\psi_{cb}. \tag{9.49}$$

For this to be satisfied for any pair of i, j,

$$\delta_{ab} \;=\; -\psi_{ab} - \sum_{c} B_{ac}\psi_{bc}. \tag{9.50}$$

This gives

$$\begin{aligned} \psi_{ab} &= -Q^{-1}_{ab}, \\ Q_{ab} &= \delta_{ab} + B_{ab}. \end{aligned} \tag{9.51}$$

Finally, this gives

$$\langle(\delta y)^2\rangle \;=\; \left\{T_i(\vec{\theta}) - T_a(\vec{\theta})\beta_{ai}\right\} D^{-1}_{ij} \left\{T_j(\vec{\theta}) - T_b(\vec{\theta})\beta_{bj}\right\}. \tag{9.52}$$

One can look at the first term in Eq. (9.52), $(T_i(\vec{\theta}) - T_a(\vec{\theta})\beta_{ai})$, and see that if θ is chosen to be one of the training points, that the resulting width will be zero. In that case, $T_a(\vec{\theta}_b)$ becomes the matrix element T_{ab}, and when multiplied by $\beta_{ai} = T^{-1}_{ab}T_{bi}$, becomes equal to T_{ai}. I.e.,

$$T_a(\vec{\theta}_b)\beta_{ai} \;= T_{ba}T^{-1}_{ac}T_{ci} = T_{bi}. \tag{9.53}$$

Inserting Eq. (9.48) and putting this all together, one has 8 terms altogether.

$$\sigma^2_E(\vec{\theta}) \;=\; \sigma^2_1 + \sigma^2_2 + \sigma^2_3 + \sigma^2_4 + \sigma^2_5 + \sigma^6_2 + \sigma^2_7 + \sigma^2_8. \tag{9.54}$$

The individual terms are

$$\begin{aligned} \sigma^2_1 &= T_i(\vec{\theta})T_i(\vec{\theta}), \\ \sigma^2_2 &= -T_a(\vec{\theta})\beta_{ai}T_i(\vec{\theta}) = -T_a(\vec{\theta})S_a(\vec{\theta}), \\ \sigma^2_3 &= \sigma^2_2, \\ \sigma^2_4 &= T_a(\vec{\theta})B_{ab}T_b(\vec{\theta}), \\ \sigma^2_5 &= T_i(\vec{\theta})\beta_{ai}\psi_{ab}\beta_{bj}T_j(\vec{\theta}) = S_a(\vec{\theta})\psi_{ab}S_b(\vec{\theta}), \\ \sigma^2_6 &= -T_a(\vec{\theta})B_{aa'}\psi_{a'b'}\beta_{b'j}T_j(\vec{\theta}) = -T_a(\vec{\theta})B_{aa'}\psi_{a'b'}S_{b'}(\vec{\theta}), \\ \sigma^2_7 &= \sigma^2_6, \\ \sigma^2_8 &= T_a(\vec{\theta})B_{aa'}\psi_{a'b'}B_{b'b}T_b(\vec{\theta}). \end{aligned} \tag{9.55}$$

Here, we have defined the quantity

$$S_a(\vec{\theta}) \ \equiv \beta_{ai} T_i(\vec{\theta}). \tag{9.56}$$

By first calculating $S_a(\vec{\theta})$, this allows one to avoid any sums over two indices i and j. One can also avoid expedite the calculations of σ_6^2 and σ_8^2 by calculating and storing

$$\begin{aligned} H_{ab}^{(6)} &= B_{aa'}\psi_{a'b}, & (9.57)\\ H_{ab}^{(8)} &= B_{aa'}\psi_{a'b'}B_{b'b}. & (9.58)\end{aligned}$$

These quantities need be calculated only once as they do not depend on $\vec{\theta}$. One then has

$$\begin{aligned} \sigma_6^2 &= -T_a(\vec{\theta})H_{ab}^{(6)}S_b(\vec{\theta}), & (9.59)\\ \sigma_8^2 &= T_a(\vec{\theta})H_{ab}^{(8)}T_b(\vec{\theta}). & (9.60)\end{aligned}$$

This avoids any $N_{\rm train}^3$ or $N_{\rm train}^4$ calculations. Thus, assuming $N_{\rm coef} >> N_{\rm train}$, the largest numeric penalty in calculating σ^2 is in the $N_{\rm train} \times N_{\rm coef}$ loop required to calculate $S_a(\vec{\theta})$ for all a.

10 Theoretical Basis of *Simplex Sampler*

Here, we imagine a spherically symmetric prior, e.g. one that is purely Gaussian, and where the parameters are scaled so that the prior distribution is invariant to rotations. If one believe the function were close to linear, a strategy would be to find $N_{\rm train} = N + 1$ points in parameter space placed far apart from another. One choice is the $N-$dimensional simplex. Examples are an equilateral triangle in two-dimensions or a tetrahedron in three dimensions. For a simplex, one places the $N_{\rm train} = N + 1$ training points at a uniform distance from the origin, r, with equal separation between each point. One can generate an $N-$dimensional simplex from an $N - 1$ dimensional arrangement. In the $N - 1$ dimensional arrangement, the points are arranged equidistant from one another using the coordinates $x_1 \cdots x_N$. The points would all be placed at a radius r_N from the origin in this system, and the separation would be d. In the $N-$dimensional system all these $N - 1$ points had coordinate $x_N = -a$. The $(N + 1)^{\rm th}$ point is then placed at position $x_1 \cdots x_N = 0$ and $x_{N+1} = Na$. This keeps the center of mass at zero. One then chooses a such that the new point is equally separated from all the other points by the same distance d,

$$\begin{aligned} d^2 &= r_{N-1}^2 + N^2a^2, & (10.1)\\ a &= \sqrt{\frac{d^2}{N^2} - r_{N-1}^2}. \end{aligned}$$

Now, each of the N points is located a distance Na away from the center of the $N-$dimensional origin. This procedure can applied iteratively to generate the vertices of the simplex.

One might also wish to use enough training points to uniquely determine the emulator in the case that the function is quadratic. There are $N(N + 1)/2$ additional points, which is exactly the number of segments connecting the $N + 1$ equally-spaced vertices of the $N-$dimensional simplex.

If placed at the midpoints of the segments, these points would be closer to the origin than the vertices. One of the simplex options is to place these points at the midpoints, then double their radii while maintaining their direction. This would result in arrays of points at two different radii, with $N+1$ points positioned at the lower radius and $N(N+1)/2$ points being placed at the larger radius.

Choosing the training points depends on prior expectation of the emulated function. The simplex choice for the first $N_{\rm train} = N+1$ points seems logical. Even if another method, such as a Gaussian process emulator is to implemented, such methods are often based on first understanding the linear behavior. The simplex strategy would seem a good way to pick the first $N+1$ training points.

One issue with the simplex is that the first set of $N+1$ training points would all be placed at the same radius. If the prior parameter distribution is uniform within an $N-$dimensional hyper cube, the training points could be rather far from the corners in that space. Issues with such priors are discussed in the next section.

10.1 The Pernicious Nature of Step-Function Priors in High Dimension

For purely Gaussian priors, one can scale the prior parameter space to be spherically symmetric. Unfortunately, that is not true for step function priors (uniform within some range). In that case the best one can do (if the priors for each parameter are independent) is to scale the parameter space such that each parameter has the constraint, $-1 < \theta_i < 1$. If the number of parameters is N, the hyper-cube has $2N$ faces and 2^N corners, a face being defined as one parameter being ± 1 while the others are zero, while a corner has each parameter either ± 1. For 10 parameters, there are 1024 corners, and for 15 parameters there are 32678 corners. Thus, it might be untenable to place a training point in each corner.

One can also see the problem with placing the training points in a spherically symmetric fashion as is done with the *Simplex Sampler*. The hyper-volume of the parameter-space hyper-cube is 2^N, whereas the volume of an $N-$dimensional hyper-sphere of radius $R=1$ is

$$V_{\rm sphere} = \Omega_N \int_0^R dr\ r^{N-1} = \Omega_N \frac{R^N}{N}. \tag{10.2}$$

The solid angle, Ω_N in N dimensions is

$$\Omega_N = \frac{2\pi^{N/2}}{\Gamma(N/2)}, \tag{10.3}$$

and after putting this together, the fraction of the hyper-cube's volume that is within the hyper-sphere is

$$\frac{V_{\rm sphere}}{V_{\rm cube}} = \left\{ \begin{array}{ll} \frac{(\pi/2)^{N/2}}{N!!}, & N = 2,4,6,8\cdots \\ \frac{(\pi/2)^{(N-1)/2}}{N!!}, & N = 3,5,7,\cdots \end{array} \right. \tag{10.4}$$

In two dimensions, the ratio is $\pi/4$, and in three dimensions it is $\pi/6$. In 10 dimensions it is 2.5×10^{-3}. For high dimensions only a small fraction of the parameter space can ever lie inside

inside a sphere used to place points. And, if the model is expensive, it may not be tenable to run the full model inside every corner.

One can also appreciate the scope of the problem by considering the radius of the corners vs. the radius of the sphere. The maximum value of $|\vec{\theta}|$ is $\sqrt{N}$. So, for 9 parameters, if the training points were all located at positions $|\vec{\theta}| < 1$, one would have to extrapolate all the way to $|\vec{\theta}| = 3$. Thus, unless the model is exceptionally smooth, one needs to devise a strategy to isolate the portion of likely parameter space using some original set of full-model runs, then augment those runs in the likely region.

A third handle for viewing the issue in N dimensions is to compare the r.m.s. radii of the hyper-sphere to that of the hyper-cube. For the cube where each side has length $2a$,

$$\left(R^{(\rm cube)}_{\rm r.m.s.}\right)^2 = a^2\frac{N}{3}. \tag{10.5}$$

whereas for a sphere of radius a,

$$\left(R^{(\rm sphere)}_{\rm r.m.s.}\right)^2 = a^2\frac{N+2}{N}. \tag{10.6}$$

The ratio of the radii is then

$$\frac{R^{(\rm sphere)}_{\rm r.m.s.}}{R^{(\rm cube)}_{\rm r.m.s.}} = \sqrt{\frac{3}{N+2}}. \tag{10.7}$$

Thus, in 10 dimensions, if the training points are placed at a distance a from the origin, the r.m.s. radii of the interior space would be half that of the entire space. Further, the r.m.s. radii of the points in the cube, $a\sqrt{N/3}$, would be about 83% higher than the radius of the training points.

Of course, these problems would be largely avoided if the number of parameters was a half dozen or fewer, or if one was confident that the function was extremely smooth. In the first two sections of this paper, the smoothness parameter, Λ, was set to a constant. There might be prior knowledge that certain parameters affect the observables only weakly. In that case, the response to these parameters can be considered as linear. This could be done by scaling those parameters so that they vary over a smaller range. If a parameter varies only between -0.1 and 0.1, that effectively applies a smoothness parameter in those directions that is ten times higher. Unfortunately, the choice of which parameters to rescale in this fashion would likely vary depending on which observable is being emulated. Because all the observables might be calculated in a full model run, one needs to identify parameters that would likely have weak response on all observables.

11 Tests of the *Smooth Emulator* using *Simplex Sampler* for Training

First, we show some results of one-parameter emulators. For a linear fit, two parameters (slope and intercept) would suffice. But because higher-rank contributions exist, there is a spread of functions is that narrows with increasing smoothness parameter as shown in Fig. 11.1. Figure 11.1 also shows

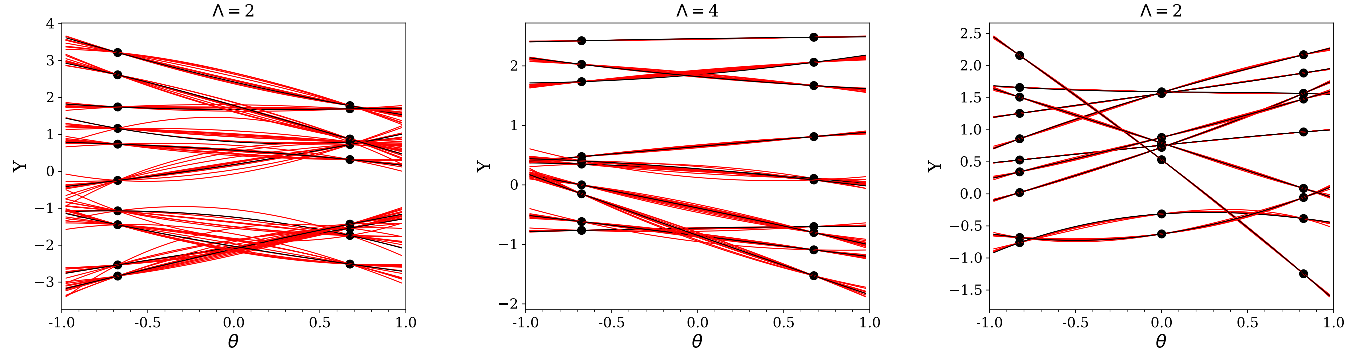

Figure 11.1: Real functions (black lines) are emulated with the *Smooth Emulator* (default settings). Ten different functions were generated sampled by the Monte Carlo method. The spread of the lines represents the uncertainty. The spread is reduced by either reducing the smoothness parameter, $\boldsymbol{\Lambda}$, or by increasing the number of sampling points.

how increasing the number of training points, from two to three, also narrows the range of possible values.

Next, we repeat the same test with six parameters. The prior distribution of parameters was uniform in the region $\boldsymbol{-1 < \theta_i < 1}$. In this case, the *Simplex Sampler* was used to choose the training points. The "real" model, $\boldsymbol{F}$, was constructed from sample smooth functions, with the coefficients generated randomly and a smoothness parameter set to three or six. The $\boldsymbol{A_{\vec{n}=0}}$ coefficient for the real model was set to zero to better accommodate viewing results. The emulator was not given that information. Twenty instances of real models were emulated. For each real model five random point in parameter space were chosen. The emulator value and its uncertainty are plotted alongside the real-model values in Fig. 11.2. The 100 comparisons show that the emulator accurately predicts the uncertainty. This is not surprising, because the emulator used the same smoothness parameter as was used to construct the real models. The width parameters, $\boldsymbol{\sigma}$, were explored stochastically, and remarkably they seem to fluctuate around the same value used to construct the real model, albeit with fluctuations of the order of 30%.

Figure 11.2 first shows the 100 comparisons for the case with the smoothness parameter, $\boldsymbol{\Lambda = 3}$. The simplest simplex form was applied, which has seven parameters, the same number one would use for a linear fit. Uncertainties were provided by finding 16 independent samplings of $\boldsymbol{A}$ coefficients and of $\boldsymbol{\sigma}$, then using the variance of the 16 samples to define the uncertainty. Variant *a* of the default emulator was used, i.e. the coefficient $\boldsymbol{A_0}$ was not given a constraining prior distribution, whereas all other coefficients were weight by a Gaussian of width $\boldsymbol{\sigma}$. In the other two panels of Fig. 11.2, the procedure was repeated but once with a higher smoothness parameter, and once with a the 28 training points, placed according to the procedure mentioned in Sec. **??**, where an additional set of points in parameter space was generated by choosing points that bisect all the lines connecting points in the original simplex, then doubling their radius. As expected, smoother functions are more easily emulated with a given number of training points, and using more training points also improves the accuracy.

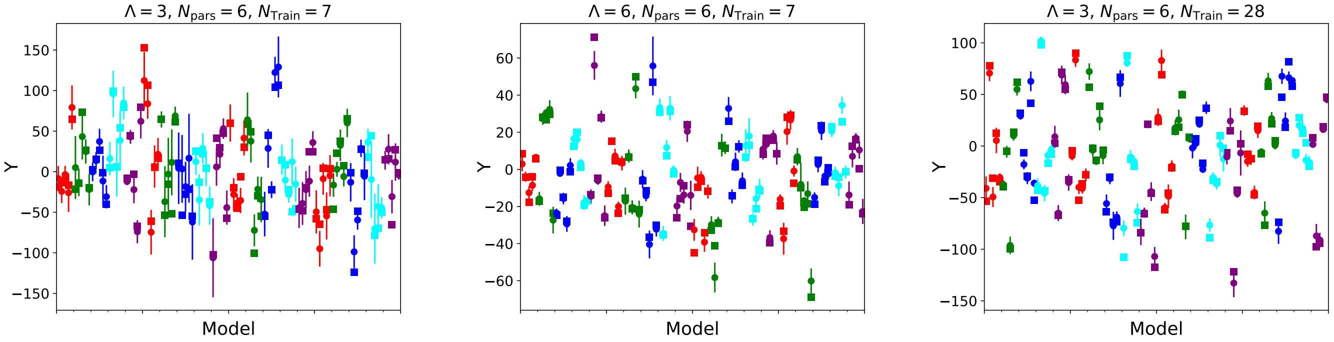

Figure 11.2: Using 20 instances of real models using six parameters, choosing 5 random points in parameter space for each model, the emulator (circles) and its uncertainty were compared to the real values (squares). Neighboring points of the same color emulated the same real model. The accuracy improves if a smoother function is considered (middle panel), or if more training points are used (right panel). Estimates of the uncertainty seem reasonable given that the uncertainties illustrated in the figure represent one standard deviation. It should be emphasized that this consistency depends critically on the fact that the emulator chose the same smoothness parameter as was used to generate the real models.