9 Underlying Theory of *Smooth Emulator*

The choice of model emulators, $\boldsymbol{E}(\vec{\boldsymbol{\theta}})$, depends on the prior understanding of the model being emulated, $\boldsymbol{M}(\vec{\boldsymbol{\theta}})$. 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:

- $\boldsymbol{E}(\vec{\boldsymbol{\theta}}_t) = \boldsymbol{M}(\vec{\boldsymbol{\theta}}_t)$ at the training points, $\vec{\boldsymbol{\theta}}_t$.
- 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, $\boldsymbol{\theta_i},\ \boldsymbol{i = 1 \cdots N}$
- 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 $\boldsymbol{E}(\vec{\boldsymbol{\theta}})$ is chosen,

$$\boldsymbol{E}(\vec{\boldsymbol{\theta}}) = \sum_{\vec{\boldsymbol{n}},\text{s.t. } \boldsymbol{K}(\vec{\boldsymbol{n}}) \leq \boldsymbol{K}_{\max}} \boldsymbol{d}_{\vec{\boldsymbol{n}}} \boldsymbol{f}_{\boldsymbol{K}(\vec{\boldsymbol{n}})}(|\vec{\boldsymbol{\theta}}|) \boldsymbol{A}_{\vec{\boldsymbol{n}}} \left(\frac{\boldsymbol{\theta_1}}{\boldsymbol{\Lambda}}\right)^{\boldsymbol{n_1}} \left(\frac{\boldsymbol{\theta_2}}{\boldsymbol{\Lambda}}\right)^{\boldsymbol{n_2}} \cdots \left(\frac{\boldsymbol{\theta_N}}{\boldsymbol{\Lambda}}\right)^{\boldsymbol{n_N}}. \tag{9.1}$$

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

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

The variance, $\boldsymbol{\sigma_K^2}$, is allowed to vary as a function of $\boldsymbol{K}$.

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

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

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

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

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

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

Choosing $\boldsymbol{f_{K=0}(0) = 1}$ and $\boldsymbol{d_{n_i=0} = 1}$, the variance of $\boldsymbol{E}$ is indeed $\boldsymbol{\sigma_0^2}$. The variance at some point $\boldsymbol{\vec{\theta} \neq 0}$ is

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

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

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

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

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

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

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

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

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

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

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

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

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

$$E(\vec{\theta}) = \sum_{\vec{n},\text{s.t. } K(\vec{n})<K_{\max}} f_K(|\vec{\theta}|) \left(\frac{K(\vec{n})!}{n_1!\cdots n_N!}\right)^{1/2} A_{\vec{n}} \left(\frac{\theta_1}{\Lambda}\right)^{n_1} \left(\frac{\theta_2}{\Lambda}\right)^{n_2} \cdots \left(\frac{\theta_N}{\Lambda}\right)^{n_N}, \tag{9.11}$$

$$P(\vec{A}_n) = \frac{1}{(2\pi\sigma_K^2)^{1/2}} e^{-|A_{\vec{n}}|^2/2\sigma_K^2}.$$

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

$$E(\vec{\theta}) = \sum_{\vec{n},K(\vec{n})\leq K_{\max}} \frac{1}{\Lambda^K} \frac{A_{\vec{n}}}{\sqrt{n_1!n_2!\cdots n_N!}} \theta_1^{n_1}\cdots\theta_N^{n_N}, \tag{9.12}$$

$$P(A_{\vec{n}}) \sim e^{-A_{\vec{n}}^2/2\sigma^2}.$$

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 $K_{\max}$ 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.

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

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

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

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 $K_{\max} \to \infty$ limit,

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

In this limit one can also see that

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

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 $K_{\max}$ 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 $x \to 1$. If such behavior is not expected, then this choice would be unreasonable.

9.3 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 $N_{\mathbf{train}}$ to be the number of training points and $N_{\mathbf{coef}}$ to be the number of coefficients, which is much larger than $N_{\mathbf{train}}$. Given the coefficients, $\vec{A}_i$ for $i > N_{\mathbf{train}}$, the first coefficients, $i = 1 \cdots N_{\mathbf{train}}$, are found by solving the linear equations for $A_a, a \leq N_{\mathbf{train}}$. For shorthand, we define the function $T_a(\vec{\theta})$,

$$E(\vec{\theta}) \;=\; \sum_{i=1}^{N_{\mathbf{coeff}}} A_i \mathcal{T}_i(\vec{\theta}), \tag{9.17}$$

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

$$T_{ai} \equiv \mathcal{T}_i(\vec{\theta}_a), \tag{9.18}$$

where $\vec{\theta}_a$ labels the training points, $0 < a \leq N_{\mathbf{train}}$, and $N_{\mathbf{train}} << N_{\mathbf{coef}}$.

9.3.1 Finding the Optimum Coefficients and Uncertainty

The goal is to find the coefficients, $A_{i>N_{\mathbf{train}}}$, that maximizes the probability given the constraints of matching the training data. If the training data are to be exactly matched, the differential probability of a given set of coefficients

$$dP(\vec{A}, \sigma_A, \Lambda) \sim \frac{1}{(2\pi\sigma_A^2)^{N_{\mathbf{coef}}/2}} \prod_{a=1}^{N_{\mathbf{train}}} \delta[y(\vec{\theta}_a) - E(\vec{\theta}_a)] \prod_{i=1}^{N_{\mathbf{coef}}} dA_i \; d\sigma_A^2 d\Lambda \exp\left\{-\frac{1}{2\sigma^2} A_i^2\right\}. \tag{9.19}$$

If the training constraints are not exact, the delta function should be replaced. This would be true if the training had some sort of point-by-point noise, such as one would have if the full-model calculation involved some sort of sampling or Monte Carlo evaluations. However, more generally one could also consider the fact that the training calculation might have some sort of error, e.g. that of missing physics. In that even the error might be correlated. If errors in the full-model calculations at points $\vec{\theta}_a$ and $\vec{\theta}_b$ are characterized some error matrix,

$$\Delta_{ab} = \langle \delta y_a \delta y_b \rangle, \tag{9.20}$$

Here, $y_a = y(\vec{\theta}_a)$, the full model value calculated at the training point a. The delta function above is replaced by

$$\begin{aligned} \prod_a \delta[y_a - E(\vec{\theta}_a)] &\to \frac{|\Delta|^{1/2}}{(2\pi)^{N_{\mathbf{train}}/2}} \exp\left\{\frac{-1}{2}(y_a - E(\vec{\theta}_a))\Delta_{ab}^{-1}(y_b - E(\vec{\theta}_b))\right\} \\ &= \int \prod_a \left(\frac{dk_a}{2\pi}\right) \exp\left\{ik_a(y_a - A_i T_i(\vec{\theta}_a)) - \frac{1}{2}k_a \Delta_{ab} k_b\right\}. \end{aligned} \tag{9.21}$$

Defining the net probability after integrating over the coefficients above, but not over σ_A^2 or Λ,

$$\mathcal{Z} \equiv \frac{dP(\sigma_A, \Lambda)}{d\sigma_A d\Lambda}, \tag{9.22}$$

the average coefficients, and correlation of coefficients, for a given $\boldsymbol{\sigma_A}$ and $\boldsymbol{\Lambda}$, can be expressed in the following form after replacing the delta function,

$$\begin{aligned}
\langle \boldsymbol{A_j} \rangle &= \boldsymbol{\mathcal{Z}^{-1}} \int \prod_i \frac{\boldsymbol{dA_i}}{\sqrt{\boldsymbol{2\pi\sigma_A^2}}}\ \boldsymbol{A_j} \exp\left[-\sum_\ell \frac{\boldsymbol{A_\ell^2}}{\boldsymbol{2\sigma_A^2}}\right] \\
&\qquad \int \prod_a \frac{\boldsymbol{dk_a}}{\boldsymbol{2\pi}} \exp\left[\boldsymbol{ik_b}(\boldsymbol{y_b} - \boldsymbol{A_i T_i}(\vec{\boldsymbol{\theta}}_b)) - \frac{\boldsymbol{1}}{\boldsymbol{2}}\boldsymbol{k_a \Delta_{ab} k_b}\right], \\
\langle \boldsymbol{A_j A_k} \rangle &= \boldsymbol{\mathcal{Z}^{-1}} \int \prod_i \frac{\boldsymbol{dA_i}}{\sqrt{\boldsymbol{2\pi\sigma_A^2}}}\ \boldsymbol{A_j A_k} \exp\left[-\frac{\boldsymbol{A_i^2}}{\boldsymbol{2\sigma_A^2}} - \frac{\boldsymbol{1}}{\boldsymbol{2}}\boldsymbol{k_a \Delta_{ab} k_b}\right] \\
&\qquad \int \prod_a \frac{\boldsymbol{dk_a}}{\boldsymbol{2\pi}} \exp\left[\boldsymbol{ik_b}(\boldsymbol{y_b} - \boldsymbol{A_i T_i}(\vec{\boldsymbol{\theta}}_b))\right], \\
\boldsymbol{\mathcal{Z}} &= \int \prod_i \frac{\boldsymbol{dA_i}}{\sqrt{\boldsymbol{2\pi\sigma_A^2}}}\ \exp\left[-\frac{\boldsymbol{A_i^2}}{\boldsymbol{2\sigma_A^2}}\right] \int \frac{\boldsymbol{dk}}{\boldsymbol{2\pi}} \exp\left[\boldsymbol{ik_b}(\boldsymbol{y_b} - \boldsymbol{A_\ell T_\ell}(\vec{\boldsymbol{\theta}}_b)) - \frac{\boldsymbol{1}}{\boldsymbol{2}}\boldsymbol{k_a \Delta_{ab} k_b}\right].
\end{aligned} \tag{9.23}$$

After completing the square, the correlation can be written as

$$\begin{aligned}
\langle \boldsymbol{A_j A_k} \rangle &= \boldsymbol{\mathcal{Z}^{-1}} \int \prod_a \frac{\boldsymbol{dk_a}}{\boldsymbol{2\pi}} e^{\boldsymbol{ik_b y_b} - \frac{1}{2}\boldsymbol{k_a \Delta_{ab} k_b}} \\
&\qquad \int \prod_i \boldsymbol{dA_i}\ \boldsymbol{A_k A_j} \exp\left[-\frac{\boldsymbol{1}}{\boldsymbol{2\sigma_A^2}}\left(\boldsymbol{A_i} + \boldsymbol{i\sigma_A^2 T_i}(\vec{\boldsymbol{\theta}}_b)\boldsymbol{k_b}\right)^{\boldsymbol{2}} - \frac{\boldsymbol{\sigma_A^2}}{\boldsymbol{2}}\left(\sum_\ell \boldsymbol{T_\ell}(\vec{\boldsymbol{\theta}}_b)\boldsymbol{k_b}\right)^{\boldsymbol{2}}\right] \\
&= \boldsymbol{\mathcal{Z}^{-1}} \int \prod_a \frac{\boldsymbol{dk_a}}{\boldsymbol{2\pi}}\ \exp\left[-\frac{\boldsymbol{\sigma_A^2}}{\boldsymbol{2}}(\boldsymbol{T_j}(\vec{\boldsymbol{\theta}}_b)\boldsymbol{k_b})^{\boldsymbol{2}} + \boldsymbol{ik_b y_b} - \frac{\boldsymbol{1}}{\boldsymbol{2}}\boldsymbol{k_a \Delta_{ab} k_b}\right] \\
&\qquad \int \prod_i \boldsymbol{dA_i}\ (\boldsymbol{A_k} - \boldsymbol{i\sigma_A^2 T_k}(\vec{\boldsymbol{\theta}}_b)\boldsymbol{k_b})(\boldsymbol{A_j} - \boldsymbol{i\sigma_A^2 T_j}(\vec{\boldsymbol{\theta}}_c)\boldsymbol{k_c}) \exp\left[-\sum_\ell \frac{\boldsymbol{A_\ell^2}}{\boldsymbol{2\sigma_A^2}}\right]
\end{aligned} \tag{9.24}$$

Similarly,

$$\begin{aligned}
\langle \boldsymbol{A_j} \rangle &= -\boldsymbol{\mathcal{Z}^{-1}} \int \prod_a \frac{\boldsymbol{dk_a}}{\boldsymbol{2\pi}}\ \exp\left[-\frac{\boldsymbol{\sigma_A^2}}{\boldsymbol{2}}(\boldsymbol{T_j}(\vec{\boldsymbol{\theta}}_b)\boldsymbol{k_b})^{\boldsymbol{2}} + \boldsymbol{ik_b y_b} - \frac{\boldsymbol{1}}{\boldsymbol{2}}\boldsymbol{k_a \Delta_{ab} k_b}\right] \\
&\qquad \int \prod_i \frac{\boldsymbol{dA_i}}{\sqrt{\boldsymbol{2\pi\sigma_A^2}}}\ \boldsymbol{i\sigma_A^2 T_j}(\vec{\boldsymbol{\theta}}_c)\boldsymbol{k_c} \exp\left[-\frac{\boldsymbol{A_\ell^2}}{\boldsymbol{2\sigma_A^2}}\right], \\
\boldsymbol{\mathcal{Z}} &= \int \prod_a \frac{\boldsymbol{dk_a}}{\boldsymbol{2\pi}}\ \exp\left[-\frac{\boldsymbol{\sigma_A^2}}{\boldsymbol{2}}(\boldsymbol{T_j}(\vec{\boldsymbol{\theta}}_b)\boldsymbol{k_b})^{\boldsymbol{2}} + \boldsymbol{ik_b y_b} - \frac{\boldsymbol{1}}{\boldsymbol{2}}\boldsymbol{k_a \Delta_{ab} k_b}\right].
\end{aligned} \tag{9.25}$$

We define the operator,

$$\boldsymbol{B_{ab}} \equiv \sum_i \boldsymbol{T_i}(\vec{\boldsymbol{\theta}}_a)\boldsymbol{T_i}(\vec{\boldsymbol{\theta}}_b) + \frac{\boldsymbol{\Delta_{ab}}}{\boldsymbol{\sigma_A^2}}. \tag{9.26}$$

Doing the integrals for $\boldsymbol{\mathcal{Z}}$,

$$\boldsymbol{\mathcal{Z}} = \boldsymbol{\sigma_A^{-N_{\text{train}}}}\ (\det \boldsymbol{B})^{-1/2} \exp\left[-\frac{\boldsymbol{1}}{\boldsymbol{2\sigma_A^2}}\boldsymbol{y_a B_{ab}^{-1} y_b}\right]. \tag{9.27}$$

One can then find the moments of the integral over $\boldsymbol{k}$,

$$\begin{aligned}\langle \boldsymbol{k_a} \rangle &= -i\partial_{y_a} \ln(\boldsymbol{\mathcal{Z}}) = \frac{\boldsymbol{i}}{\boldsymbol{\sigma_A^2}} \boldsymbol{B_{ab}^{-1} y_b}, \\ \langle \boldsymbol{\delta k_a \delta k_b} \rangle &= -\partial_{y_a}\partial_{y_b} \ln(\boldsymbol{\mathcal{Z}}) = \frac{\mathbf{1}}{\boldsymbol{\sigma_A^2}} \boldsymbol{B_{ab}^{-1}}.\end{aligned} \tag{9.28}$$

The moments are then:

$$\begin{aligned}\langle \boldsymbol{A_j} \rangle &= \boldsymbol{T_j(\vec{\theta}_a) B_{ab}^{-1} y_b}, \\ \langle \boldsymbol{\delta A_j \delta A_k} \rangle &= \boldsymbol{\delta_{jk}\sigma_A^2 - \sigma_A^2 B_{ab}^{-1} T_j(\vec{\theta}_a) T_k(\vec{\theta}_b)}.\end{aligned} \tag{9.29}$$

The observables and their fluctuations are then:

$$\begin{aligned}\langle \boldsymbol{E(\vec{\theta})} \rangle &= \langle \boldsymbol{A_j} \rangle \boldsymbol{T_j(\vec{\theta})}, \\ \langle \boldsymbol{\delta E(\vec{\theta})^2} \rangle &= \langle \boldsymbol{\delta A_j \delta A_k} \rangle \boldsymbol{T_j(\vec{\theta}) T_k(\vec{\theta})}.\end{aligned} \tag{9.30}$$

After definining

$$\begin{aligned}\boldsymbol{\chi_a} &\equiv \boldsymbol{B_{ab}^{-1} y_b}, \\ \boldsymbol{S_a(\vec{\theta})} &\equiv \boldsymbol{T_j(\vec{\theta}) T_j(\vec{\theta}_a)},\end{aligned} \tag{9.31}$$

the mean and fluctuation can be written as

$$\begin{aligned}\langle \boldsymbol{E(\vec{\theta})} \rangle &= \boldsymbol{\chi_a S_a(\vec{\theta})}, \\ \langle \boldsymbol{\delta E(\vec{\theta})^2} \rangle &= \boldsymbol{\sigma_A^2 T_j(\vec{\theta}) T_j(\vec{\theta}) - \sigma_A^2 S_a(\vec{\theta}) B_{ab}^{-1} S_b(\vec{\theta})}.\end{aligned} \tag{9.32}$$

Whereas $\langle \boldsymbol{E(\vec{\theta})} \rangle$ depends linearly on the training values, $\boldsymbol{y_{a=1,N_{\text{train}}}}$, the emulator's uncertainty, $\langle \boldsymbol{\delta E(\vec{\theta})^2} \rangle$ depends only on the location of the training points.

If one sets $\boldsymbol{\vec{\theta} = \vec{\theta}_c}$, one of the training points, and if $\boldsymbol{\Delta = 0}$, then

$$\begin{aligned}\boldsymbol{S_a(\vec{\theta}_c)} &= \boldsymbol{B_{ac}}, \\ \boldsymbol{T_j(\vec{\theta}_c) T_j(\vec{\theta}_c)} &= \boldsymbol{B_{cc}}, \ \textbf{no sum over } \boldsymbol{c},\end{aligned} \tag{9.33}$$

and

$$\langle \boldsymbol{\delta E(\vec{\theta}_c)^2} \rangle = \boldsymbol{\sigma_A^2 B_{cc} - \sigma_A^2 B_{ca} B_{ab}^{-1} B_{bc} = 0}. \tag{9.34}$$

Thus, there is no uncertainty when evaluated at the training points. When $\boldsymbol{\Delta \neq 0}$, the uncertainty at a training point, $\boldsymbol{\vec{\theta}_c}$ is

$$\langle \boldsymbol{\delta E(\vec{\theta}_c)^2} \rangle = \boldsymbol{\Delta_{cc} - \frac{1}{\sigma_A^2} \Delta_{ca} B_{ab}^{-1} \Delta_{bc}}, \quad \textbf{(no sum over } \boldsymbol{c}\textbf{)}. \tag{9.35}$$

If $\boldsymbol{\Delta}$ is small, the uncertainty of the emulator is simple given by $\boldsymbol{\Delta}$, however it is somewhat reduced for larger $\boldsymbol{\Delta}$. Because the matrices $\boldsymbol{T_i(\vec{\theta}_a) T_i(\vec{\theta}_b)}$ and $\boldsymbol{\Delta_{ab}}$ are both positive definite, the uncertainty squared above is manifestly positive.

Before considering a specific position, $\vec{\boldsymbol{\theta}}$, one first calculates and stores the vector $\boldsymbol{\chi}$ and the matrix $\boldsymbol{B^{-1}}$. This involves inverting a $\boldsymbol{N_{\mathbf{train}} \times N_{\mathbf{train}}}$ matrix. Then, when considering a specific $\vec{\boldsymbol{\theta}}$, one calculates $\boldsymbol{S_a(\vec{\theta})}$ for each $\boldsymbol{a}$. This involves calculating a double sum with $\boldsymbol{N_{\mathbf{train}} \times N_{\mathbf{coef}}}$ elements. Once these are stored, calculating $\boldsymbol{\langle \delta y^2 \rangle}$ involves calculating a double sum with an additional $\boldsymbol{N_{\mathbf{train}} \times N_{\mathbf{train}}}$ elements. Thus, calculating $\boldsymbol{S_a(\vec{\theta})}$ is where the bulk of the calculation occurs. Because $\boldsymbol{N_{\mathbf{train}} \lesssim 100}$ and $\boldsymbol{N_{\mathbf{coef}}}$ might be of in the tens of thousands (depending on the number of parameters and maximum rank), one should be able to calculate $\boldsymbol{\langle y \rangle}$ and $\boldsymbol{\langle \delta y^2 \rangle}$ in much less than a second. Nonetheless, when performing an MCMC calculation, one might calculate for millions of values of $\vec{\boldsymbol{\theta}}$, which can push the calculation into the ranges of minutes or even hours if one has a dozen parameters or more.

The calculations above could have been performed finding the coefficients to maximize the probability, $\boldsymbol{P(\vec{A})}$, rather than the mean coefficients, but the answer is exactly the same.

9.3.2 Choosing the Variance for the Coefficients and the Smoothness Parameter

Here, we find the optimum choice for $\boldsymbol{\sigma_A}$, the variance of the coefficient in the absence of training data, in the previous section. For the derivations of $\boldsymbol{\langle y \rangle}$ and $\boldsymbol{\langle \delta y^2 \rangle}$ in the previous section, the expression in Eq. (9.29) for $\boldsymbol{\langle A_i \rangle}$ has no dependence on $\boldsymbol{\sigma_A}$, which sets the variance of the coefficients. Further, the expression for $\boldsymbol{\langle \delta A_i^2 \rangle}$ does not depend on the model training values, $\boldsymbol{y_{a=1,N_{\mathrm{train}}}}$. The net probability from Eq. (9.36) was found to be in Eq. (9.19) to be

$$\int \prod_i \boldsymbol{dA_i P(\vec{A}, \sigma^2, \Lambda)} = \boldsymbol{\mathcal{Z}} \tag{9.36}$$

$$= \boldsymbol{\sigma_A^{-N_{\mathrm{train}}}} \left(\det \boldsymbol{B}\right)^{1/2} \exp\left[-\frac{\boldsymbol{1}}{\boldsymbol{2\sigma_A^2}} \boldsymbol{y_a B^{-1}_{ab} y_b}\right].$$

Our goal here is to find the value of $\boldsymbol{\sigma_A}$ that maximizes $\boldsymbol{\mathcal{Z}}$. For the case where $\boldsymbol{\Delta = 0}$, one can find $\boldsymbol{\sigma_A}$ rather easily. In that case, setting the derivative w.r.t. $\boldsymbol{\sigma_A^2}$ equal to zero,

$$\boldsymbol{0 = \frac{N_{\mathrm{train}}}{\sigma_A^2} \mathcal{Z} - \frac{\mathcal{Z}}{\sigma_A^4} y_a B^{-1}_{ab} y_b,} \tag{9.37}$$

$$\boldsymbol{\sigma_A^2 = \frac{1}{N_{\mathrm{train}}} y_a B^{-1}_{ab} y_b.} \tag{9.38}$$

However, the situation is becomes more complicated if $\boldsymbol{\Delta \neq 0}$. Fortunately, our tests show that the value of $\boldsymbol{\sigma_A^2}$ is only very slightly changed by random uncertainties if those uncertainties are less than 10 percent of $\boldsymbol{\sigma_A}$. For that reason, *Smooth Emulator* ignores $\boldsymbol{\Delta_{ab}}$ while choosing $\boldsymbol{\sigma_A}$. Tests (see last subsection) showed that the appropriate value of $\boldsymbol{\sigma_A}$ can usually be found to within 10%.

Choosing the smoothness parameter, $\boldsymbol{\Lambda}$, is unfortunately not simple. Given the lack of an analytic solution, one can simply plot $\boldsymbol{\ln(Z)}$ vs $\boldsymbol{\Lambda}$. However, it was found (as shown in the final subsection here) that afater using a function for the surrogate full-model characterized by a known $\boldsymbol{\Lambda}$, the value at which $\boldsymbol{\ln(Z)}$ is minimized can vary significantly from the true value. Though this method reproduces the correct value if averaged over many possible surrogate functions, it varies too much on a per-emulation basis to be useful. For example, for some instances of sample functions, it was

found that $\mathbf{ln(Z)}$ would be maximized for $\mathbf{\Lambda < 1.5}$, even when the surrogate-model function was generated with $\mathbf{\Lambda = 3}$.

Our conclusion (see final sub-section) regarding the ability *Smooth Emulator* to determine $\boldsymbol{\sigma_A}$ and $\mathbf{\Lambda}$ is that $\boldsymbol{\sigma_A}$ can be found quite robustly using the simple analytic form in Eq. (**??**), but that the determination of $\mathbf{\Lambda}$ is questionable. Fortunately, the predicted values of the emulated function are not strongly sensitive to $\mathbf{\Lambda}$, but the estimated emulator uncertainties are somewhat sensitive. Thus, one can plot $\mathbf{ln(Z)}$ vs $\mathbf{\Lambda}$ to get some insight, but the result should be taken with caution. It might make more sense to set $\mathbf{\Lambda}$ to some guess, perhaps 2.5, then just live with the knowledge that this might not be the best choice. If one can perform numerous additional full-model runs at points not used for training, one can choose $\mathbf{\Lambda}$ such that the uncertainties are large enough to capture 68% of the points within one standard deviation. From the experience of trying many functions, it would seem that optimizing $\mathbf{\Lambda}$ in this way can improve the uncertainty estimates by approximately 50%. I.e. changing $\mathbf{\Lambda}$ by factors of 2 or 4, tends to change the estimated uncertainties by tens of percent.

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 $\boldsymbol{N_{train} = N + 1}$ points in parameter space placed far apart from another. One choice is the $\boldsymbol{N}-$dimensional simplex. Examples are an equilateral triangle in two-dimensions or a tetrahedron in three dimensions. For a simplex, one places the $\boldsymbol{N_{train} = N + 1}$ training points at a uniform distance from the origin, $\boldsymbol{r}$, with equal separation between each point. One can generate an $\boldsymbol{N}-$dimensional simplex from an $\boldsymbol{N - 1}$ dimensional arrangement. In the $\boldsymbol{N - 1}$ dimensional arrangement, the points are arranged equidistant from one another using the coordinates $\boldsymbol{x_1 \cdots x_N}$. The points would all be placed at a radius $\boldsymbol{r_N}$ from the origin in this system, and the separation would be $\boldsymbol{d}$. In the $\boldsymbol{N}-$dimensional system all these $\boldsymbol{N-1}$ points had coordinate $\boldsymbol{x_N = -a}$. The $\boldsymbol{(N + 1)^{th}}$ point is then placed at position $\boldsymbol{x_1 \cdots x_N = 0}$ and $\boldsymbol{x_{N+1} = Na}$. This keeps the center of mass at zero. One then chooses $\boldsymbol{a}$ such that the new point is equally separated from all the other points by the same distance $\boldsymbol{d}$,

$$\boldsymbol{d^2 = r_{N-1}^2 + N^2 a^2}, \tag{10.1}$$
$$\boldsymbol{a = \sqrt{\frac{d^2}{N^2} - r_{N-1}^2}}.$$

Now, each of the $\boldsymbol{N}$ points is located a distance $\boldsymbol{Na}$ away from the center of the $\boldsymbol{N}-$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 $\boldsymbol{N(N + 1)/2}$ additional points, which is exactly the number of segments connecting the $\boldsymbol{N + 1}$ equally-spaced vertices of the $\boldsymbol{N}-$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 expand their radii while maintaining their direction. This would result in arrays of points at two different radii, with $\boldsymbol{N + 1}$ points positioned at the lower radius and $\boldsymbol{N(N + 1)/2}$ points being placed at the larger radius.

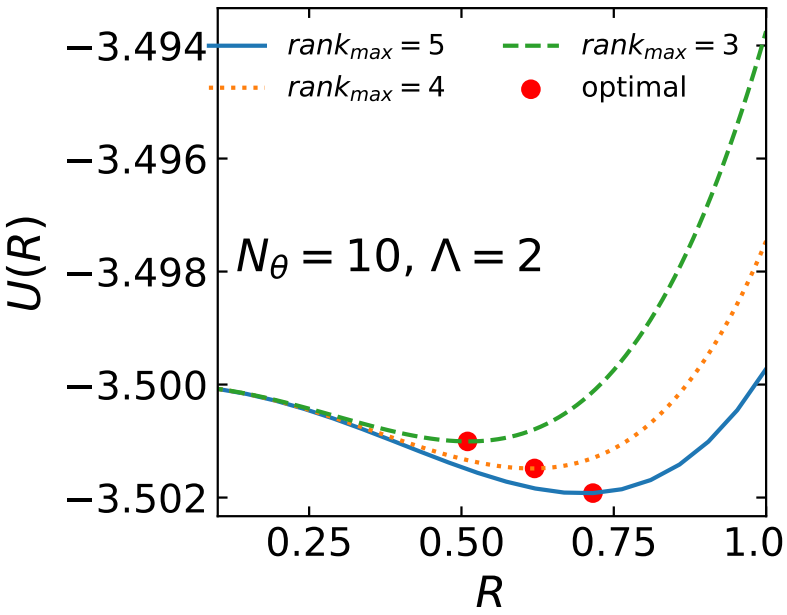

Figure 10.1: The integrated uncertainty for training with points set in a simplex geometry is found to depend only weakly on the radius. A Gaussian prior was assumed. The weak dependence was used to justify using a rather ad-hoc choice of radii for the simplex points, or as in the case of type 2, a second radius for the bisecting points.

From symmetry considerations, this method should produce either an extrema of effectiveness in regards to angular positioning (assuming Gaussian priors, which are rotationally invariant). I.e. any repositioning of a training point by a small differential $\boldsymbol{d\vec{\theta}}$ should not change the effectiveness if that differential does not change the radius of the point. However, one can expand all the radii of all the inner-positioned simplex, or all the radii of the other points, without changing the symmetry. Thus, the radii must be chosen to maximize the constraining power of the training data. To that end, a study was pursued where the overall integrated uncertainty was minimized as a function of the radius. Only the case of the simple simplex was considered, and only for the case of Gaussian priors. The minimum was found to depend on the maximum rank of the emulator, the smoothness parameter, and the number of observables. However, the minimum was always very broad, as shown in Fig. 10.1. For that reason, the radii were chosen rather ad hoc:

- Simple Simplex ($\boldsymbol{N+1}$ points, Type 1):
  The radii of each point was chosen to be $\boldsymbol{1-1/(N+1)}$.
- Simplex plus bisecting points ($\boldsymbol{N+1}$ inner points plus $\boldsymbol{N(N+1)}$ outer points:
  The outer points were positioned at a radius $\boldsymbol{1-1/(N+1)}$, while the inner points were positioned at a radius which was smaller by a factor of $\boldsymbol{1/\sqrt{2}}$.

One issue with the simplex is that the first set of $\boldsymbol{N+1}$ training points would all be placed at the same radius. If the prior parameter distribution is uniform within an $\boldsymbol{N}-$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, $\boldsymbol{-1 < \theta_i < 1}$. If the number of parameters is $\boldsymbol{N}$, the hyper-cube has $\boldsymbol{2N}$ faces and $\boldsymbol{2^N}$ corners, a face being defined as one parameter being $\boldsymbol{\pm 1}$ while the others are zero, while a corner has each parameter either $\boldsymbol{\pm 1}$. 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 $\boldsymbol{2^N}$, whereas the volume of an $\boldsymbol{N}-$dimensional hyper-sphere of radius $\boldsymbol{R = 1}$ is

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

The solid angle, $\boldsymbol{\Omega_N}$ in $\boldsymbol{N}$ dimensions is

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

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

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

In two dimensions, the ratio is $\boldsymbol{\pi/4}$, and in three dimensions it is $\boldsymbol{\pi/6}$. In 10 dimensions it is $\boldsymbol{2.5 \times 10^{-3}}$. 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 $\boldsymbol{|\vec{\theta}|}$ is $\boldsymbol{\sqrt{N}}$. So, for 9 parameters, if the training points were all located at positions $\boldsymbol{|\vec{\theta}| < 1}$, one would have to extrapolate all the way to $\boldsymbol{|\vec{\theta}| = 3}$. 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 $\boldsymbol{N}$ 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 $\boldsymbol{2a}$,

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

whereas for a sphere of radius $\boldsymbol{a}$,

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

The ratio of the radii is then

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

Thus, in 10 dimensions, if the training points are placed at a distance $\boldsymbol{a}$ 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, $\boldsymbol{a}\sqrt{\boldsymbol{N}/\mathbf{3}}$, 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, $\boldsymbol{\Lambda}$, 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 $\mathbf{-0.1}$ and $\mathbf{0.1}$, 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 *Smooth Emulator*

Here, we show several results for testing *Smooth Emulator*. For these tests we consider a surrogate full-model generated from Taylor expansions with randomized coefficients, exactly according to the form assumed by the emulator. These functions are characterized by a smoothness parameter, $\boldsymbol{\Lambda}$, a variance for the coefficients, $\boldsymbol{\sigma_A}$, and a maximum rank. If the emulator works correctly, 68% of the discrepancies in emulated functions, compared to the true model, should fall within one standard deviation. Performing the same test as described in the tutorial, Fig. 11.1 compares emulator predictions to 50 different surrogate models. Each surrogate model was constructed according to Taylor expansions with $\boldsymbol{\Lambda} = \mathbf{3}$, a maximum rank of 4, and $\boldsymbol{\sigma_A} = \mathbf{100}$. Although only 50 such models are illustrated here, this was attempted with 5000 models, and it was found that the percentage of predictions within one sigma was within a few tenths of a percent of the 68% expected, thus validating both the emulator's prediction of both its value and its estimate of its uncertainty.

Figure 11.2 shows the average of surrogate-model predictions $\mathbf{ln}(\boldsymbol{Z})$, $\boldsymbol{\sigma_A}$ and the percentage of predictions within one sigma over 500 instances. These are shown as a function of the emulator's $\boldsymbol{\Lambda}$. Because the surrogate models were constructed with $\boldsymbol{\Lambda} = \mathbf{3}$, the value of $\mathbf{ln}(\boldsymbol{Z})$ should be a maximum there. Further, for $\boldsymbol{\Lambda} = \mathbf{3}$, the extracted value of $\boldsymbol{\sigma_A}$ should be 100, and 68% of emulator predictions should be within the emulator's uncertainty estimate. The emulator passes all these tests beautifully, but as can be seen, whereas the prediction of $\boldsymbol{\sigma_A}$ is remarkably accurate, the value of $\mathbf{ln}(\boldsymbol{Z})$ is quite uncertain. Thus, finding $\boldsymbol{\Lambda}$ for a given emulator might be problematic. Figure 11.3 shows $\mathbf{ln}(\boldsymbol{Z})$ for several individual models. This was repeated for 5, 10 and 15 parameters. For smaller numbers of parameters, using $\mathbf{ln}(\boldsymbol{Z})$ vs $\boldsymbol{\Lambda}$ to determine $\boldsymbol{\Lambda}$ is especially problematic. Thus, the User might want to simply fix $\boldsymbol{\Lambda}$ at some value, e.g 2.5, and live with the consequences.

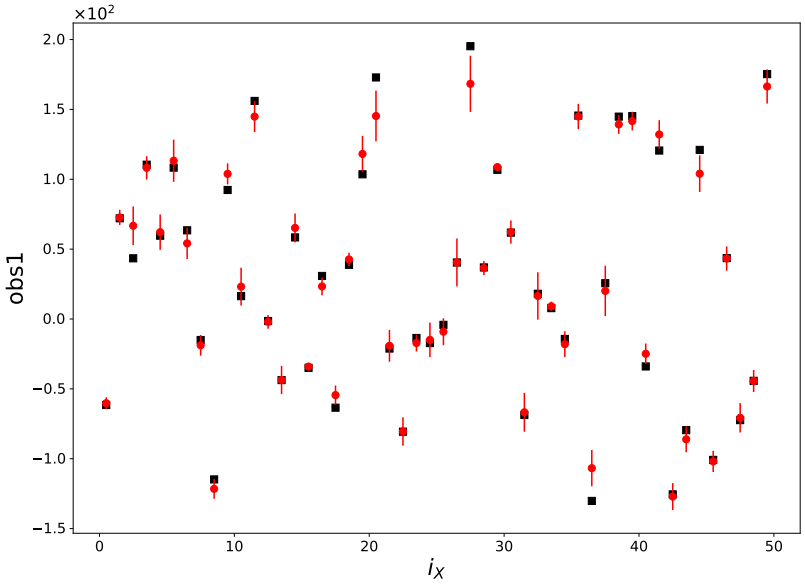

Figure 11.1: Emulator predictions are compared to a surrogate model with Taylor expansions based on $\boldsymbol{\Lambda = 3}$ and $\boldsymbol{\sigma_A = 100}$. As can be seen approximately 68% of the predictions were within one sigma of the true model, validating the emulator, both in regards to its predicted value and its uncertainty estimate.

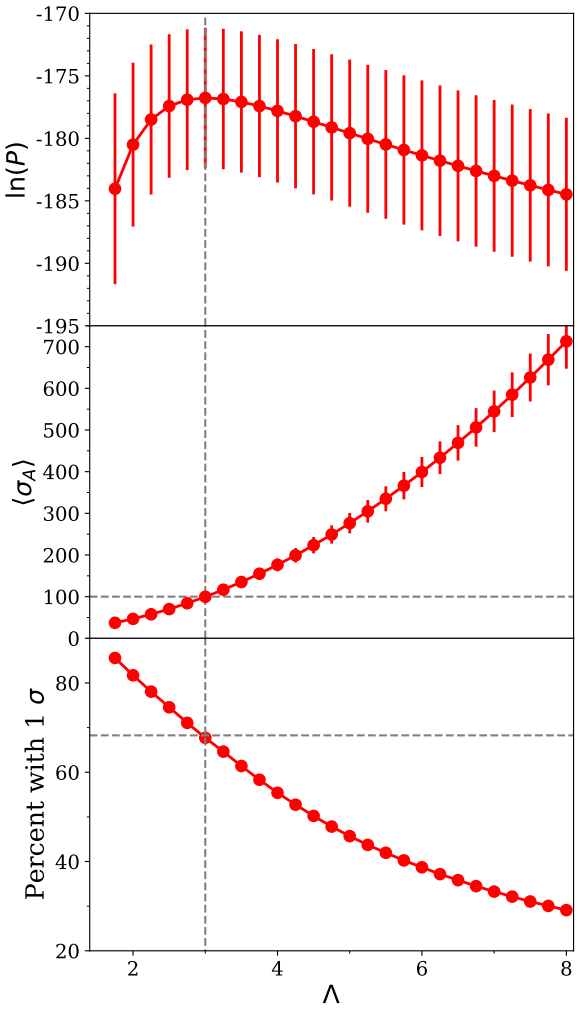

Figure 11.2: A ensemble of 5000 surrogate models, each generated with $\mathbf{\Lambda = 3}$ an $\boldsymbol{\sigma_A = 100}$ were emulated. The extracted values of $\mathbf{ln(Z)}$ an $\boldsymbol{\sigma_A}$, along with the percentage of emulator predictions withing the emulator's uncertainty were plotted agains the value of $\mathbf{\Lambda}$ assumed for the emulator. Indeed, for $\mathbf{\Lambda = 3}$, $\mathbf{ln(Z)}$ is at a maximum, $\boldsymbol{\sigma_A = 100}$ and 68% of the predictions are within the estimated uncertainty. Unfortunately, $\mathbf{ln(Z)}$ has a high instance-to-instance variability.

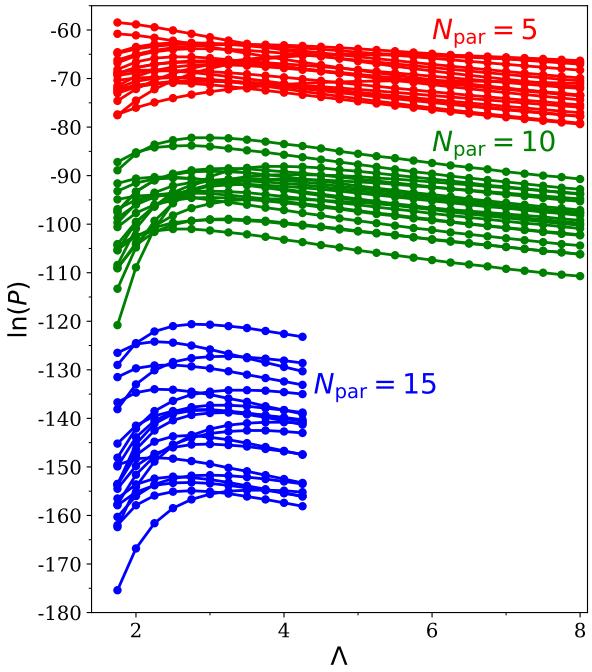

Figure 11.3: For 20 instances of the surrogate function, $\mathbf{ln(Z)}$ is plotted vs. $\mathbf{\Lambda}$. Although if one were to average over thousands of surrogate models, one could extract $\mathbf{\Lambda}$ accurately, the extraction is rather inaccurate on a model-by-model basis. This is especially true if the model has smaller numbers of parameters.