Project 2: Stochastic expansion and approximation methods

Due Wednesday 9 November 2022, end of day

This project focuses on the use of stochastic expansions and approximation methods to solve a partial differential equation (PDE) with uncertain coefficients and boundary conditions. You will implement several of these methods and assess their accuracy and convergence, i.e., how error depends on a variety of truncation orders and on sample size. You will also have the opportunity to experiment with alternative ways of computing global sensitivity indices

1 Part 1: stochastic building blocks and Monte Carlo simulations

1.1 Model (familiar from Project #1)

Consider a stochastic elliptic equation on a one-dimensional spatial domain:

$$\frac{\partial}{\partial x}\left(k(x,\omega)\frac{\partial u(x,\omega)}{\partial x}\right) = -s(x), \qquad x \in \mathcal{D} = [0,1], \tag{1}$$

with a deterministic source term $s(x)$, a deterministic Dirichlet boundary condition at $x = 1$,

$$u(1,\omega) = u_r,$$

and a random Neumann condition at $x = 0$,

$$k(x,\omega)\frac{\partial u}{\partial x}\bigg|_{x=0} = -F(\omega).$$

The diffusivity $k(x,\omega)$ and solution $u(x,\omega)$ are stochastic processes defined on $\mathcal{D}\times\Omega$, where $(\Omega,\mathcal{U},\mathbb{P})$ is a probability space. $F(\omega)$ is defined on the same probability space. This stochastic elliptic equation can model a host of physical phenomena, ranging from heat conduction in a heterogeneous material (where u is proportional to temperature) to fluid flow in a porous medium (where u denotes pressure and k is proportional to permeability).

The MATLAB script we provided with Project #1, `diffusioneqn.m` (or any Python/Julia adaptation of it), can be used to solve this equation. Due to the different specification of k, however (see below), an analytical solution is no longer tractable.

1.2 Parameters (slightly different than in Project #1)

Parameters and boundary conditions for equation (1) are specified as follows:

- Let $Y(x,\omega)$ be a stationary Gaussian process with covariance kernel

$$C(x_1,x_2) = \sigma_Y^2 \exp\left(-\frac{1}{p}\left(\frac{|x_1-x_2|}{L}\right)^p\right)$$

  and mean $\mathbb{E}[Y(x,\omega)] = \mu_Y$. Then the diffusivity $k(x,\omega) \equiv \exp(Y(x,\omega))$. Use the following parameter values: mean $\mu_Y = 1.0$, correlation length $L = 0.3$, variance $\sigma_Y^2 = 0.3$, and exponent $p = 1.0$.

- The flux F is normally distributed with mean $\mu_F = -1.0$ and variance $\sigma_F^2 = 0.2$. F and k are independent.

- The Dirichlet datum is $u_r = 1$.

- The source term is spatially uniform: $s(x) = 5$.

1.3 Part 1 questions

These questions focus on representing the stochastic building blocks of the problem and on establishing a baseline Monte Carlo solution.

(a) Construct a polynomial chaos expansion of the diffusivity field $k(x,\omega)$:

  (i) First develop a Karhunen-Loève (K-L) expansion of $Y(x,\omega)$. Quantify the error resulting from truncation of this expansion at a finite stochastic dimension n. If you use a Nyström method to approximate the K-L eigenfunctions and eigenvalues, you will need a Gauss-Legendre quadrature rule (Gaussian quadrature rule for uniform weight on the domain $\mathcal{D}$). The script `qrule.m`, provided with the project, can compute such rules.

  (ii) Use the K-L expansion of Y to compute a total-degree polynomial chaos expansion of $k(x,\omega) = \exp(Y(x,\omega))$.

  (iii) Numerically explore the convergence of this polynomial chaos expansion with respect to stochastic dimension n *and* polynomial degree p. Describe any qualitative differences in the realizations that result from differerent truncations in n and p, and also evaluate some quantitative metrics for convergence. (Think about what is a suitable/comprehensive L^2-type error!)

(b) Use Monte Carlo simulation to generate realizations of the solution of the stochastic elliptic equation. In particular, plot the probability density of the solution $u(x_i,\omega)$ at selected points $x_i \in D$. Calculate and plot the mean of the solution field $\mu_u(x) =$

$\mathbb{E}[u(x,\omega)]$ and the covariance of the solution field $C_u(x,x')$. Also, just plot several realizations of the solution field.

You can drive your Monte Carlo simulations using either the untruncated $Y(x,\omega)$ (i.e., no truncation of its representation beyond that implicit in the grid used to represent the solution of the ODE), a truncated K-L expansion of $Y(x,\omega)$, or the polynomial chaos expansion of $k(x,\omega)$. Try all three, and comment on how errors in the representation of the stochastic coefficient affect the stochastic solution $u(x,\omega)$.

(c) Consider u at $x = 0.5$. Which factor contributes more to the variance of the solution at this point: the flux F or the diffusivity k? Answer this question quantitatively by calculating *main* and *total-effect* global *sensitivity indices* for $u(x = 0.5)$ with respect to k and F, using Saltelli's method.

1.4 Hints and suggestions for Part 1

1. To compute the polynomial chaos expansion of the log-normal process k, recall that we derived (analytically) the Hermite chaos expansion of a log-normal random *variable* in class. You will need to generalize this derivation to multiple stochastic dimensions.

2. In computing the polynomial chaos expansion with Hermite polynomials, take care to avoid confusion between the "physicist" Hermite polynomials (orthogonal with respect to a weight of $w(x) \propto \exp(-x^2)$) and the "probabilist" Hermite polynomials (orthogonal with respect to a weight $w(x) \propto \exp(-x^2/2)$). We recommend using the probabilist version.

3. It may be useful, in solving part (a-ii) and in preparation for Part 2, to have a code that enumerates all multi-indices of a total-degree polynomial expansion, i.e., all elements of the set $\{\boldsymbol{\alpha} \in \mathbb{N}_0^n : \|\boldsymbol{\alpha}\|_1 \leq p\}$, for arbitarary n and p. We have posted such a code, along with one that enumerates multi-indices for a full tensor-product expansion: $\{\boldsymbol{\alpha} \in \mathbb{N}_0^n : \|\boldsymbol{\alpha}\|_\infty \leq p\}$. Just open `multi_indices.m` and see the MATLAB scripts therein. In particular, run `example.m`. You will not need to modify the other scripts.

4. To compute the global sensitivity indices in part (c) with any reasonable accuracy, you may need a *large number* of samples. Go big!

2 Part 2: solving the stochastic PDE

In the first half of the project, you investigated the Karhunen-Loève representation of the log-diffusivity field $Y(x,\omega)$ and the polynomial chaos expansion of the diffusivity field $k(x,\omega)$. You also characterized the solution of the stochastic elliptic PDE using Monte Carlo and performed some global sensitivity analysis. In Part 2 of the project, you will solve the same PDE using a least-squares polynomial approximation.

In the notation below, we will assume that the K-L expansion of Y has been truncated after $d-1=n$ stochastic dimensions. Adding an additional stochastic dimension for the uncertain boundary condition, the problem will have a total of d stochastic dimensions, corresponding to independent input random variables $\xi_1,\ldots,\xi_d$. Since Y is a Gaussian field and the boundary condition is Gaussian, these input random variables can all be made *standard* Gaussian (i.e., $\xi_i \sim N(0,1)$).

2.1 Part 2 questions

2.1.1 Least-squares polynomial approximation

The goal of this approach is to construct a polynomial approximation of the solution field $u(x_*,\omega)$, at any point $x_* \in D$, non-intrusively—i.e., using only deterministic solutions of the PDE at selected (random) realizations of the inputs.

(a) Fix a point $x_* \in D$. Approximate the solution $u(x_*,\xi_1,\ldots,\xi_d)$ of the stochastic elliptic equation using a least-squares polynomial approach. In particular, use a total-degree p polynomial space and Monte Carlo sampling from the input d-dimensional standard Gaussian measure to construct your approximation.

(b) Explore the convergence of your solution with respect to expansion degree p and the number of Monte Carlo samples m. What happens when m is too small? Roughly what values of m do you need to start obtaining stable and accurate results?

(c) *(Enrichment problem; try this if you have time!)* The Gaussian measure (or, in general, the measure defining the orthogonal polynomial family) is actually *not* the optimal measure from which to sample when constructing least squares approximations. Recent work shows that another measure, called the pluripotential equilibrium measure, can yield more accurate and stable results, particularly for high polynomial degree. See [A. Narayan, J. Jakeman, T. Zhou (2017), "A Christoffel function weighted least squares algorithm for collocation approximations," *Mathematics of Computation*, 86: 1913–1947].

Implement the algorithm for *weighted* least squares polynomial approximation described in the paper. In particular, see Algorithm 3 and the fourth row of Table 2. Comment on the results, compared to part (b), particularly for large p (and if necessary, smaller d).

For additional background and examples, have a look at [L. Guo, A. Narayan, T. Zhou (2020), "Constructing least-squares polynomial approximations," *SIAM Review*, 62: 483–508], which is also posted on our Canvas site.

2.1.2 Post-processing the solution

(a) Using the polynomial expansions obtained above, plot the probability density of the solution $u(x_i,\omega)$ at selected points $x_i \in D$. Calculate and plot the mean of the solution

field $\mu_u(x) = \mathbb{E}[u(x,\omega)]$ and the covariance of the solution field $C_u(x,x')$. Also plot several realizations of the solution field. Compare these to the Monte Carlo results you obtained in Part 1.

(b) Consider the Karhunen-Loève expansion of the solution field $u(x,\omega)$. Compute (numerically) the *eigenvalues* of this K-L expansion and compare their decay to that of the eigenvalues of the K-L expansion of $Y(x,\omega)$. Plot both sets of eigenvalues in a single figure, where each set is normalized by its largest eigenvalue, i.e., λ_i^Y/λ_1^Y and λ_i^u/λ_1^u. Comment on the difference between the two spectra and what may be causing it.

(c) Consider u at $x = 0.5$. Which factor contributes more to the variance of the solution at this point: the flux F or the diffusivity k? Answer this question quantitatively by calculating *main* and *total-effect* global *sensitivity indices* for $u(x = 0.5)$, just as you did in Part 1, but this time using the polynomial chaos expansion directly.

2.2 Hints and suggestions for Part 2

1. There is no need to introduce the polynomial expansion of $k = \exp(Y)$ from Part 1 here; consider that a separate exercise. In this part, just use the K-L expansion $Y_n(x,\omega)$ that you developed in Part 1 and set $k(x,\omega) = \exp(Y_n(x,\omega))$ directly. Coefficients of the K-L expansion are effectively *inputs* to the deterministic solver that yields the solution field $u(x)$ for any value of the diffusivity coeficient.

2. Truncation of the K-L expansion of Y is a matter of taste, but for the purposes of solving the stochastic elliptic equation here—particularly since we are using a non-adaptive expansion—it is probably best to stay with fewer than ten dimensions, perhaps more like five.

3. In computing the least-squares approximation, try normalizing the polynomials, i.e., making them not just orthogonal but *orthonormal*, before using them to assemble the Vandermonde matrix.

4. Compute the global sensitivity indices in part 2.1.2(c) *after* computing the polynomial chaos expansion of $u(x = 0.5,\omega)$. With the polynomial chaos expansion in hand, the sensitivity indices can be derived analytically. Then you can compare to those you obtained in Part 1.

5. To compute the probability density functions in part 2.1.2(a), it is recommended that you just sample the polynomial expansion. (Use many samples, since it's cheap!) To compute the mean and covariance, you could also sample, but it's more accurate/elegant to evaluate these quantities directly from the expansion coefficients.

6. At each spatial location x_i, the polynomial coefficients of you expansion of $u(x_i,\omega)$ can be calculated independently. But, if you'd like to be more efficient in practice, since

your deterministic solution of the elliptic PDE will yield the entire field $u(x)$ for any setting of the uncertain input parameters, you can combine or vectorize operations and develop the polynomial approximation at multiple $\{x_i\}$ simultaneously.

3 Appendix: sparse grid pseudospectral approximation

One alternative to least-squares polynomial approximation, which we discussed extensively in lecture, is pseudospectral approximation embedded in a sparse grid construction. Though it can be more efficient, the implementation of this method is considerably more involved. *For those who would like to try*, this appendix presents an **alternative version of Section 2.1.1** that you can choose to pursue instead. We recommend that you embark on this alternative path only if you have sufficient time! Section 2.1.2 remains the same, regardless of how you built your polynomial approximation.

The following questions will build up the sparse grid polynomial approximation step-by-step, starting with a generic *full tensor* approximation, and then combining these full tensor approximations according to a Smolyak rule.

3.1 Alternative Section 2.1.1 questions

(a) Fix a point $x_* \in D$. Construct a *full tensor pseudospectral approximation* of $u(x_*, \xi_1, \ldots, \xi_d)$, i.e.,

$$u(x_*, \xi_1, \ldots, \xi_d) \approx S_{\mathbf{m}}^{(1:d)} u(x_*, \xi_1, \ldots, \xi_d) = \left[S_{m_1}^{(1)} \otimes \cdots \otimes S_{m_d}^{(d)}\right] u(x_*, \xi_1, \ldots, \xi_d),$$

truncated according to a multi-index $\mathbf{m} \coloneqq (m_1, \ldots, m_d) \in \mathbb{N}_0^d$. Here, m_i denotes the maximum polynomial degree in the input ξ_i. Make sure that your routine works (in principle) for any choice of $\mathbf{m}$, particularly choices where many of the components of $\mathbf{m}$ are zero.

Choosing nonzero values of m_i for many inputs will lead to an expensive computation, so don't try too a high degree in more than a handful of coordinates. We will remedy this problem in the next part. But make your code as general as possible.

(b) Now we will construct a *sparse* pseudospectral approximation. Use a Smolyak approach to develop a sparse polynomial chaos expansion of $u(x_*, \omega)$ in higher stochastic dimensions. Use an isotropic truncation $\|\boldsymbol{\ell}\|_1 \leq P$, where the multi-index $\boldsymbol{\ell}$ denotes the *level* of the constituent full tensor pseudospectral approximation. (See hints below for a discussion of the relationship between $\boldsymbol{\ell}$ and $\mathbf{m}$.)

(c) Examine the convergence of your solution with respect to the truncation level P of the sparse pseudospectral approximation.

(d) Comment on the number of deterministic solutions required.

3.2 Hints for sparse grid pseudospectral approximation

1. To perform Gaussian quadrature, the posted script `qrule.m` should be quite useful. But pay attention to the appropriate scaling/normalization of the weight function—i.e., take care to modify the points and weights generated by `qrule` so that they account for the "probabilist" weight of $w(x) \propto \exp(-x^2/2)$ rather than the "physicist" weight of $w(x) \propto \exp(-x^2)$. This just requires a change of variables.

2. To develop your pseudospectral approximation, a first choice is to use a one-dimensional quadrature rule (for the standard Gaussian measure, as noted above) and to tensorize it. A Gauss-Hermite rule is the natural choice, but it is not "nested." For more efficiency in practice, one would like to use a nested rule, as we will discuss in lecture. A nested rule for Gaussian weight is provided Genz & Keister, *J. Comp. Appl. Math.,* 1996. We suggest first trying a regular Gauss-Hermite rule, and experimenting with the nested rule for fun later, only if you have time!

3. The goal of question 3.1(a) is to develop a full-tensor pseudospectral approximation operator, where quadrature rule and the polynomial degree are appropriately *matched* and indexed by a single parameter m_i for each dimension. With Gaussian quadrature, the number of points in each univariate quadrature rule should be $m_i + 1$, so that an integrand of degree $2m_i$ can be integrated exactly. This avoids "internal aliasing" in the pseudospectral approximation and should make each full tensor product approximation equivalent to a polynomial interpolation.

4. In question 3.1(b), we wish to combine full tensor approximations of different "levels." There are many ways to define the "level" here. The simplest is just to set $\boldsymbol{\ell} = \mathbf{m}$, and thus the Smolyak rule will combine a sequence of approximations that differ only by one polynomial degree. But this is not the most efficient choice.

   One better choice is $m_i = 2\ell_i$, for $\ell_i = 0, 1, 2, \ldots$. (Think about why this might be better.) Another (often even more efficient) choice is the exponential growth $m_i = 2^{\ell_i+1} - 2$, again for $\ell_i = 0, 1, 2, \ldots$.

   As you experiment with the Smolyak approach, you may get the sense that making sparse grids efficient in practice requires a careful *caching* of function values that have already been computed, to avoid redundant computations! This is particularly true if one uses a nested quadrature rule. It's great to notice this—and caching is part of any practical sparse grid code—but don't worry about implementing such a caching strategy for this project.

5. Isotropic and non-adaptive truncation of the Smolyak approximation is sufficient for this project.