TO BE OR NOT TO BE INTRUSIVE?
THE SOLUTION OF PARAMETRIC AND STOCHASTIC
EQUATIONS—THE “PLAIN VANILLA” GALERKIN CASE

LOÏC GIRALDI†, ALEXANDER LITVINENKO‡, DISHI LIU§, HERMANN G. MATTHIES¶, AND ANTHONY NOUY†

Abstract. In parametric equations—stochastic equations are a special case—one may want to approximate the solution such that it is easy to evaluate its dependence on the parameters. Interpolation in the parameters is an obvious possibility—in this context often labeled as a collocation method. In the frequent situation where one has a “solver” for a given fixed parameter value, this may be used “nonintrusively” as a black-box component to compute the solution at all the interpolation points independently of each other. By extension, all other methods, and especially simple Galerkin methods, which produce some kind of coupled system, are often classed as “intrusive.” We show how, for such “plain vanilla” Galerkin formulations, one may solve the coupled system in a nonintrusive way, and even the simplest form of block-solver has comparable efficiency. This opens at least two avenues for possible speed-up: first, to benefit from the coupling in the iteration by using more sophisticated block-solvers and, second, the possibility of nonintrusive successive rank-one updates as in the proper generalized decomposition (PGD).

Key words. parametric problems, stochastic equation, uncertainty quantification, Galerkin approximation, coupled system, nonintrusive computation

AMS subject classifications. 65M70, 65J15, 60H25, 65D30, 34B08, 35B30

DOI. 10.1137/130942802

1. Introduction. Many problems depend on parameters, which may be a finite set of numerical values, or mathematically more complicated objects such as, for example, processes or fields. We address the situation where we have an equation which depends on parameters. A special case of such parametric problems are stochastic equations, where the parameters are elements from a probability space. One common way to represent this dependence on parameters is by evaluating the state (or solution) of the system under investigation for different values of the parameters. Particularly in the stochastic context, this “sampling” is a common procedure. But often one wants to evaluate the solution quickly for a new set of parameters which has not been sampled. In this situation it may be advantageous to express the parameter-dependent solution with an approximation which allows for rapid evaluation of the solution or functionals thereof—so-called quantities of interest (QoI)—dependent on the parameters. Such approximations are also called proxy or surrogate models, response functions, or emulators. This last term was chosen so as to contrast with simulator, which is the original solver for the full equation. Such approximations are

*Submitted to the journal’s Methods and Algorithms for Scientific Computing section October 24, 2013; accepted for publication (in revised form) July 7, 2014; published electronically November 20, 2014. This work was partly supported by the Deutsche Forschungsgemeinschaft (DFG) through the SFB 880.
http://www.siam.org/journals/sisc/36-6/94280.html
†École Centrale de Nantes, GeM, UMR CNRS 6183, LUNAM Université, 44321 Nantes, France (loic.giraldi@cc-nantes.fr, anthony.nouy@cc-nantes.fr).
‡KAUST, Thuwal 23955, Saudi Arabia (alexander.litvinenko@kaust.edu.sa).
§Institute of Aerodynamics and Flow Control, German Aerospace Center (DLR), 38108 Braunschweig, Germany (dishi.liu@dlr.de).
¶Corresponding author. Institute of Scientific Computing, Department of Computer Science, TU Braunschweig, 38092 Braunschweig, Germany (wire@tu-bs.de).
used in several fields, notably optimization and uncertainty quantification, where in
the latter case the parameters are random variables and one deals with stochastic
equations. All these methods may be seen as functional approximations, namely, rep-
resentations of the solution by an “easily computable” function of the parameters, as
opposed to pure samples.

The most obvious methods of approximation used are based on interpolation and
in this context are often labeled as collocation methods. In this case it is usually
assumed that the parameters are in some subdomain of a manifold, usually simply just
in some finite-dimensional vector space, and the interpolation is often performed on
sparse grids [5, 2, 26, 55, 34]. This process normally gives the approximation (inter-
polant) as a finite linear combination of some basis functions used for the interpolation
(often global multivariate polynomials [57], or piecewise polynomials [3, 53], or meth-
ods based on radial basis functions, kriging, or neural networks) or approximations
using reproducing kernel methods [54].

Another approach is to again choose a similar finite set of basis functions, but
rather than interpolation use some other projection onto the subspace spanned by
these functions. Usually this will involve minimizing some norm of the difference
between the true parametric solution and the approximation, and in many cases this
norm will be induced by an inner product, often in the form of an integral with
respect to (w.r.t.) some measure. In the case of stochastic equations this will be the
underlying probability measure. These integrals in turn may be numerically evaluated
through quadrature formulas. One common choice is to use Monte Carlo or quasi
Monte Carlo methods, but another possibility is to use sparse Smolyak or adaptive
grids [47, 27, 48, 32, 28]. In any case the integrand, part of which is the parametric
solution, has to be evaluated at a finite number of parameter values. Such methods
are sometimes called pseudospectral projections, or regression solutions, or discrete
projections [12, 15, 43, 44, 11, 49, 7, 31, 6, 50].

In the frequent situation where one has a “solver” for the equation for a given
parameter value, i.e., a software component or a program, it is evident that this can be
used to independently solve for all the parameter values, which subsequently may be
used either for the interpolation or in the quadrature for the projection. Such methods
are therefore uncoupled for each parameter value and obviously allow the use of the
original solver. Therefore, they additionally often carry the label “nonintrusive.”
Without much argument, all other methods are almost always labeled as “intrusive,”
meaning that one cannot use the original solver; see, e.g., [21, 30, 51, 20, 19]. We
want to show here that this is not necessarily the case.

Like most methods which are based on the solution at discrete parameter values,
the so-called nonintrusive methods mentioned above “forget” the original equation,
i.e., the fact that the approximation has to satisfy the parametric equation. This is
generally the state of affairs when using a proxy model in the domain of optimization.
On the other hand, methods which try to ensure that the approximation satisfies
the parametric equation as well as possible are often based on a Rayleigh–Ritz or
Galerkin type of “ansatz,” which leads to a coupled system for the unknown coefficients
[29, 38, 56, 3, 39, 25, 52, 36, 13]. This is often taken as an indication that the
original solver cannot be used, i.e., that these methods are “intrusive.” But in many
circumstances these methods may as well be used in a nonintrusive fashion. Although
there are some publications concerning special cases of nonintrusive Galerkin-like
methods [1, 14, 16], this has not been widely recognized as a general possibility. A
kind of in-between possibility is the so-called reduced basis method; see [8, 9] for
recent expositions. Here a new basis for the parametric solution is built from solves at particular parameter values, but the “interpolation” is achieved by a Galerkin projection onto the spanned subspace. This method also establishes a connection between emulators or proxy models and reduced order models, another field that has a strong computational influence on the problems considered here. As an example one may mention [17], but we will not pursue this topic further here.

In some publications collocation (interpolation) methods have been compared with Galerkin formulations. In [22] the result was that for the same number of unknowns, the Galerkin method is substantially more accurate than the collocation method. Put in another way, this means that for a required accuracy, far fewer unknowns are needed for the Galerkin method. In [4] the conclusion was that for analytic dependence on the parameters and carefully designed collocation points, the two methods offer comparable accuracy.

When comparing different methods, one has to distinguish at least two things. One is the potential of the method to achieve a certain accuracy with a given number of terms, the stability of the method, and the asymptotic behavior of these relations. The other is the actual computation to be carried out, the numerical effort involved, and, in our case especially, the question of whether this can be done in a nonintrusive way.

We will return to these questions in a bit more detail in subsection 3.1 and subsection 4.4, so some short remarks may suffice here: the collocation (interpolation) methods and the projection methods (pseudospectral or discrete) project the solution on an a priori chosen subspace. Collocation (interpolation) is not an orthogonal projection, and it is well known that this may lead to stability problems. Projection methods, especially pseudospectral projections, use an orthogonal projection of the solution, and thus enjoy higher stability, although the inner product used in regression methods is often not related properly to the original inner product, which, in the case of stochastic problems, is induced by the probability measure. As these methods disregard the parametric equation, it is not easy to establish a posteriori error bounds. Experience suggests that Galerkin orthogonality is very useful for obtaining such error bounds. This is available in Galerkin-like methods, which thus offer the possibility of adaptivity and algorithms which produce an approximation with guaranteed error bounds. These methods do not project the solution but the residuum of the parametric equation. They also typically do not use samples of the solution, but lead to more complex equations. The reduced basis method [8, 9] also falls into this class, even though it uses point-evaluations or, in more recent work, other linear functionals of the solution.

Collocation (interpolation) and projection methods require the solution of uncoupled equations for each sampling point. Galerkin-like methods, on the other hand, usually lead to coupled equations. Although at first sight this looks like a disadvantage, it can actually be turned into an advantage. The uncoupled solution methods do not allow for the transmission of information about the convergence behavior from one sample to another. This is possible by sophisticated block-solvers (see, e.g., [35, 41]), which therefore may allow the acceleration of the solution procedure. This will not be pursued here, but we will show that even the simplest solver—nonlinear block-Jacobi—for the coupled system is comparable in effort to projection methods, both theoretically and in terms of a little example.

Thus in some ways the difference between collocation/interpolation or projection/regression and Galerkin-like methods is that the latter use the information from the
parametric equation, whereas the former do not. Thus Galerkin-like methods need the mathematical model which is embodied by the parametric equation. This additional information may be useful for error bounds and adaptivity on one hand, but on the other hand it may also be put to use to accelerate the actual solution process.

Another important reason for the interest in Galerkin like methods are recent developments for low-rank separated approximations [24, 10, 33, 45, 18, 46, 23, 19] of parametric or stochastic equations, which are based on the minimization of a least squares or similar functional, and naturally lead to Galerkin-type equations. Although it is important to show that these cases and other low-rank approximation methods (see, e.g., [42]) can also be dealt with in a nonintrusive manner, here we concentrate on the “plain vanilla,” i.e., standard, Galerkin case. Nonintrusive computation of separated approximations will be investigated elsewhere.

Most of the literature cited so far is concerned with the case of stochastic equations, and although these are a special case of parametric equations, the methods and techniques used there may be used in the wider context of general parametric equations; see [40] for a synopsis of these connections with such general parametric problems.

The question of whether a method is intrusive or not is often very important in practice. The “solver” (for a single parameter value) may contain much specialized domain specific knowledge, and may therefore represent quite a valuable investment of effort. In the case when the method is labeled intrusive, it may seem like the whole process and effort of producing a solver, now for the coupled Galerkin system, would have to be repeated again. Therefore, in many cases the wish to re-use existing software guides the choice of method. It will be shown that such a nonintrusive approach is also possible for the Galerkin formulation.

A method for a parametric problem will be considered intrusive here if one has to modify the original software to solve the parametric problem. Thus it turns out that the question of whether a method is intrusive or not hinges on what kind of interface one has to the software, and is thus a software-engineering question. Most often it is possible to compute not only the solution for a certain parameter value, but also the residuum or a “preconditioned” residuum, given a “trial solution.” This usually means doing one iteration with the solver instead of iterating all the way to convergence. This kind of interface will be assumed here, and we will show that it can be used without any change to solve the Galerkin equations.

The plan for the rest of the paper is as follows. In section 2 following we introduce the notation and assumptions for the parametric problem. In section 3 we introduce the Galerkin approximation, describe alternative formulations, and prove the convergence and speed of a basic block-Jacobi algorithm for the coupled Bubnov–Galerkin system. In section 4 it is shown how the residual in the iteration may be computed nonintrusively, mainly via numerical integration. The behavior of the modified iterates is analyzed, and it is shown that they accumulate in the vicinity of the solution. In section 5 a small numerical example is investigated which shows how the nonintrusive computation works and confirms the theoretical predictions.

2. Parametric problems. To be more specific, let us consider the following situation: we are investigating some physical system which is modeled by an equation for its state $u \in U$, which is assumed to be a Hilbert space with inner product $\langle \cdot, \cdot \rangle_U$ for the sake of simplicity:

$$A(p; u) = f(p).$$
Here a mapping $A : \mathcal{U} \to \mathcal{U}^*$ is an operator modeling the physics of the system, and $f \in \mathcal{U}^*$ is some external influence (action/excitation/loading). The model depends on some parameters $p \in \mathcal{P}$. In many cases (1) is the abstract formulation of a partial differential equation. Although the following development could be carried out in this Hilbert space or in an even more general setting, as no specifically “finite-dimensional” arguments are used, assume for simplicity that we are dealing with a model on a finite-dimensional space with $N := \dim \mathcal{U} < \infty$, e.g., a partial differential equation after discretization. We identify $\mathcal{U}$ and $\mathcal{U}^*$, and if needed we will use an orthonormal basis $\{v_n\}_{n=1}^N$, i.e., $\text{span}\{v_n\}_{n=1}^N = \mathcal{U}$ and $(v_n, v_m)_\mathcal{U} = \delta_{n,m}$, the Kronecker-$\delta$, again to simplify arguments.

Assume that for all $p \in \mathcal{P}$, (1) is a well-posed problem. This means that $A$ as a mapping $u \mapsto A(p; u)$ for a fixed $p$ is bijective and continuously invertible, i.e., for each $p$ and $f$ it has a unique solution, which will be denoted by $u^*(p)$, such that for all $p : A(p; u^*(p)) = f(p)$.

Although this will not be needed here, let us remark that if the map $A$ were also differentiable w.r.t. $u$ with partial derivative denoted by $D_u A$, well-posedness would imply that this partial derivative is nonsingular and also continuously invertible. Assuming a differentiable structure on the set $\mathcal{P}$, one may invoke a version of the implicit function theorem, which, given the partial derivatives $D_u A$ and $D_p f$, provides the derivative of the state $u$ w.r.t. $p$ as $D_p u = [D_u A]^{-1}(D_p f - D_p A)$. This and possibly higher derivatives may be directly used in the approximation of $u^*(p)$, as well as for a priori bounds. These topics will not be pursued further here.

Further assume that we are also given an iterative solver—convergent for all values of $p$—which generates successive iterates for $k = 0, \ldots$.


$$(2) \quad u^{(k+1)}(p) = S(p; u^{(k)}(p), R(p; u^{(k)}(p))), \quad \text{with } u^{(k)}(p) \to u^*(p),$$

where $S$ is one cycle of the solver which may also depend on the iteration counter $k$, $u^{(0)}$ is some starting vector, and $R(p; u^{(k)}(p))$ is the residuum of (1),

$$R(u^{(k)}) := R(p; u^{(k)}(p)) := f(p) - A(p; u^{(k)}).$$

Obviously, when the residuum vanishes—$R(p; u^*(p)) = 0$—the mapping $S$ has a fixed point $u^*(p) = S(p; u^*(p), 0)$.

This mapping $S$ is the mathematical formalization of the software interface we will be assuming in order to derive a nonintrusive Galerkin method; i.e., we will assume that the mapping $S$ is applied to its inputs with one invocation of the “solver.”

In the iteration of (2) we may set $u^{(k+1)} = u^{(k)} + \Delta u^{(k)}$ with

$$\Delta u^{(k)} := S(p; u^{(k)}, R(p; u^{(k)})) - u^{(k)},$$

and usually

$$P(\Delta u^{(k)}) = R(p; u^{(k)}),$$

so that in (2), $S(p; u^{(k)}) = u^{(k)} + P^{-1}(R(p; u^{(k)}))$, where by slight abuse of notation we have shortened the list of arguments. Here $P$ is some preconditioner, which may depend on $p$, the iteration counter $k$, and the current iterate $u^{(k)}$, e.g., in Newton’s method $P = D_u A(p; u^{(k)})$. In any case, we assume that for all arguments the map $P$ is linear in $\Delta u$ and nonsingular. The iteration corresponding to a normal solve for a particular value of $p$ then is given in Algorithm 2.1.

We will assume additionally that the iteration converges at least linearly; i.e., one has $\|\Delta u^{(k+1)}(p)\| \leq \varrho(p) \|\Delta u^{(k)}(p)\|\|u\|$, with $\varrho(p) < 1$. For the convergence analysis to follow later we will assume that the convergence factors or Lipschitz constants $\varrho(p)$
Algorithm 2.1. Iteration of (2).

Start with some initial guess $u^{(0)}$
$k ← 0$

while no convergence do

%comment: the global iteration loop%

Compute $\Delta u^{(k)}$ according to (4) or (5)
$u^{(k+1)} ← u^{(k)} + \Delta u^{(k)}$
$k ← k + 1$

end while

are uniformly bounded for all values of $p ∈ P$ by a constant strictly less than unity, i.e., $\rho(p) ≤ \rho^* < 1$. Another way of saying this is that for all $u, v ∈ U$ and $p ∈ P$ the iterator $S$ in (2) is uniformly Lipschitz continuous with Lipschitz constant $\rho^* < 1$, i.e., a strict contraction:

\begin{equation}
\|S(p; u(p), R(p; u(p))) − S(p; v(p), R(p; v(p)))\|_U ≤ \rho^* \|u(p) − v(p)\|_U.
\end{equation}

One may recall from Banach’s fixed point theorem that this provides us with the a posteriori error bounds

\begin{equation}
\|u^*(p) − u^{(k+1)}(p)\|_U ≤ \frac{\rho^*}{1 − \rho^*} \|\Delta u^{(k)}(p)\|_U,
\end{equation}

whereas the satisfaction of the equation may be measured by the magnitude of $\|R(p; u^{(k)})\|_U$.

3. Galerkin approximation of parametric dependence. To describe the dependence of $u$ on the parameters $p$, one would like to approximate $u^*(p)$ in the following fashion:

\begin{equation}
u^*(p) ≈ u_{\mathcal{I}}(p) = \sum_{\alpha ∈ \mathcal{I}} u_\alpha \psi_\alpha(p),\end{equation}

where $u_\alpha ∈ U$ are vector coefficients to be determined, and $\psi_\alpha$ are linearly independent functions, a basis for $Q_\mathcal{I} := \text{span}\{\psi_\alpha\}_{\alpha ∈ \mathcal{I}} ⊂ \mathbb{R}^P$, the Galerkin subspace of parametric “ansatz” functions, and $\mathcal{I}$ is some finite set of (multi-)indices of cardinality $M := |\mathcal{I}|$. Often the set $\mathcal{I}$ has no canonical order, but for the purpose of computation later we will assume that some particular ordering has been chosen.

If we take the ansatz (8) and insert it into (1), the residuum (3) will usually not vanish for all $p$, as the linear combinations of basis functions $\{\psi_\alpha\}_{\alpha ∈ \mathcal{I}}$ cannot match all possible parametric variations of $u(p)$.

3.1. The Galerkin equations for the residual. The Galerkin method determines the unknown coefficients $u_\alpha$ in (8) through weighted residuals. This is achieved by choosing a set of linearly independent test or weighting functions $\{\varphi_\beta\}_{\beta ∈ \mathcal{I}} ⊂ \mathbb{R}^P$ and requiring that for all $\beta ∈ \mathcal{I}$,

\begin{equation}0 = G_{\mathcal{Q}}(\varphi_\beta(\cdot)R(\cdot; u_{\mathcal{I}})) ∈ U.
\end{equation}

The residuum $R(p; u_{\mathcal{I}}(p))$ in the argument of the linear Galerkin projector $G_{\mathcal{Q}}$ is a parametric function, and such functions may be represented by a sum $R(p; u_{\mathcal{I}}(p)) =
\( \sum_{n} \phi_{n}(p)v_{n} \) with \( \phi_{n} \in \mathbb{R}^{P} \). Hence the projector is defined by requiring that for scalar functions \( \psi, \phi \in \mathcal{Q} \subseteq \mathbb{R}^{P} \) and a vector \( v \in \mathcal{U} \) one has

\[
\forall w \in \mathcal{U} : \quad \langle G_{\mathcal{Q}}(\phi(\cdot)\psi(\cdot) v), w \rangle_{\mathcal{U}} = \langle \phi, \psi \rangle_{\mathcal{Q}} \langle v, w \rangle_{\mathcal{U}},
\]

where \( \langle \cdot, \cdot \rangle_{\mathcal{Q}} \) is a duality pairing or inner product on a subspace \( \mathcal{Q} \) of the scalar functions, and from this, \( G_{\mathcal{Q}} \) can be extended by linearity:

\[
G_{\mathcal{Q}}(\varphi_{\beta} R(\cdot; u_{I})) = \sum_{n} G_{\mathcal{Q}}(\varphi_{\beta} \phi_{n} v_{n}) = \sum_{n} \langle \varphi_{\beta}, \phi_{n} \rangle_{\mathcal{Q}} v_{n}.
\]

It is easy to see that this definition is independent of the particular representation of the parametric function.

If \( \mathcal{P} \) is a measure space with measure \( \mu \), then that pairing often is \( \langle \phi, \psi \rangle_{\mathcal{Q}} = \int_{\mathcal{P}} \phi(p)\psi(p) \mu(dp) \). And if additionally \( \mu(\mathcal{P}) = 1 \), such that \( \mathcal{P} \) may be considered as a probability space with expectation operator \( \mathbb{E}(\phi) = \int_{\mathcal{P}} \phi(p) \mu(dp) \), then \( \langle \phi, \psi \rangle_{\mathcal{Q}} = \mathbb{E}(\phi \psi) \). Observe that a sum like \( \sum_{j} w_{j}\phi(p_{j})\psi(p_{j}) \) with weights \( w_{j} \) is a special form of such an integral. To be a bit more general we would allow \( \langle \phi, \psi \rangle_{\mathcal{Q}} = \int_{\mathcal{P} \times \mathcal{P}} \kappa(p,q)\phi(p)\psi(q) \mu(dp)\mu(dq) \), where \( \kappa \) is a symmetric positive definite kernel. What is important for what is to follow, and what we want to assume from now on, is that the pairing is given by some integral, and we will assume the form \( \langle \phi, \psi \rangle_{\mathcal{Q}} = \int_{\mathcal{P}} \phi(p)\psi(p) \mu(dp) \) for the sake of simplicity. In that case the Galerkin projector \( G_{\mathcal{Q}}(\varphi_{\beta} R(\cdot; u_{I})) = \int_{\mathcal{P}} \varphi(p)_{\beta} R(p; u_{I}) \mu(dp) \in \mathcal{U} \) is given as a Bochner integral.

The basis \( \{ \psi_{\alpha} \}_{\alpha \in \mathcal{I}} \) in (8) determines and represents the Galerkin subspace \( \mathcal{Q}_{I} := \text{span}\{ \psi_{\alpha} \}_{\alpha \in \mathcal{I}} \subseteq \mathcal{Q} \), which is responsible for the approximation properties, whereas the set \( \{ \varphi_{\beta} \}_{\beta \in \mathcal{I}} \) determines the projection onto that subspace which is important for the stability of the procedure, as the projection is orthogonal to \( \mathcal{Q}_{I} := \text{span}\{ \varphi_{\beta} \}_{\beta \in \mathcal{I}} \). Often one takes \( \varphi_{\beta} = \psi_{\beta} \) and hence \( \mathcal{Q} = \mathcal{Q}_{I} \), and this is then commonly called the Bubnov–Galerkin method, whereas in the general case \( \mathcal{Q}_{I} \neq \mathcal{Q} \) one speaks of the Petrov–Galerkin method.

Explicitly writing down (9), one obtains for all \( \beta \),

\[
0 = G_{\mathcal{Q}} \left( \varphi_{\beta}(\cdot) \left( f(\cdot) - A \left( \sum_{\alpha \in \mathcal{I}} u_{\alpha} \psi_{\alpha}(\cdot) \right) \right) \right) \in \mathcal{U}.
\]

It is important to recognize that (11) is a—usually coupled—system of \( M \times N \) equations for the unknown vectors \( u_{\alpha} \in \mathcal{U} \), as \( M = \dim \mathcal{Q}_{I} \) and \( N = \dim \mathcal{U} \). These equations look sufficiently different from (1), so that the common wisdom is that the solution of (11) requires new software and new methods, and that the solver (2) is of no use here. As a change or rewrite of the existing software seems to be necessary, the resulting methods are often labeled “intrusive.”

Observe that if one chooses \( \varphi_{\beta}(p) = \delta_{\beta}(p) = \delta(p - p_{\beta}) \)—the delta-“function” associated to the duality pairing \( \langle \cdot, \cdot \rangle_{\mathcal{Q}} \) (i.e., \( \langle \delta_{\beta}, \phi \rangle_{\mathcal{Q}} = \phi(p_{\beta}) \))—where the \( p_{\beta} \) are distinct points in \( \mathcal{P} \) in (17), this becomes for all \( \beta \),

\[
0 = G_{\mathcal{Q}}(\delta_{\beta} R(\cdot; u_{I})) = R(p_{\beta}; u_{I}(p_{\beta}))
\]

\[
= f(p_{\beta}) - A \left( p_{\beta}; \sum_{\alpha \in \mathcal{I}} u_{\alpha} \psi_{\alpha}(p_{\beta}) \right) = f(p_{\beta}) - A(p_{\beta}; u_{\beta}) \in \mathcal{U},
\]
where the last of these equalities holds only in the case when the basis \{\psi_\alpha\} satisfies the Kronecker-\delta property \psi_\alpha(p_\beta) = \delta_\alpha_\beta, as then \psi_\beta = u_I(p_\beta). In this latter case these are \(M\) uncoupled equations, each of size \(N = \text{dim} \mathcal{U}\), and they have for each \(p_\beta\) the form (1)—we have recovered the collocation method which, independently for each \(p_\beta\), computes \(u_\beta\), using the solver (2). Such a method then is obviously nonintrusive, as the original software may be used. Thus this is often the method of choice, as often there is considerable investment in the software which performs (2), which one would like to re-use. Unfortunately this choice is very rigid in regard to the subspace \(Q_I\) and the projection orthogonal to \(Q_I\).

We believe that this is a false alternative and that the distinction is not between intrusive or nonintrusive, but between coupled or uncoupled. Furthermore, and more importantly, we want to show that also in the more general case of a coupled system, like in (11), the original solver (2) may be put to good use. This will be achieved by making (2) the starting point, instead of (1) or (3). Such coupled iterations also arise, for example, from multiphysics problems, and these coupled iterations can also be solved by what is called a partitioned approach (see, e.g., [35, 41]), which is the equivalent of nonintrusive here. Quite a few different variants of global partitioned iterations are possible [41]; we look only at some of the simplest variants, as the point here is only to dispel the myth about intrusiveness.

Recall from the introduction, section 1, the short discussion about the different methods to solve such a parametric problem. In all methods a finite-dimensional subspace \(Q_I = \text{span}\{\psi_\alpha\} \subset Q\) is chosen, and this subspace determines how well the solution \(u^*(p)\) can be approximated as a function of \(p \in \mathcal{P}\).

In the collocation/interpolation method and in the projection (pseudospectral or regression) method the solution \(u^*(p)\) is projected onto \(Q_I \otimes \mathcal{U}\) to give \(u_I(p)\). The choice of \(Q_I = \text{span}\{\varphi_\beta\}\) determines the kind of projection \(\tilde{\Pi}(u^*(p)) = u_I(p)\), as it is required to be orthogonal to \(Q_I\):\n
\[
\forall \phi \in Q_I : \quad G_Q(\phi(\cdot)(u^*(\cdot) - u_I(\cdot))) = G_Q(\phi(\cdot)(u^*(\cdot) - \tilde{\Pi}(u^*(\cdot)))) = 0.
\]

The norm of the projection \(\tilde{\Pi}\) is a measure of stability of the whole procedure. As \(\tilde{\Pi}\) is the identity on \(Q_I \otimes \mathcal{U}\), the norm satisfies \(\|\tilde{\Pi}\| \geq 1\); it depends on the angle between the subspaces \(Q_I\) and \(Q_T\).

The collocation/interpolation method chooses points \(p_\beta\), and with that \(Q_C := \hat{Q}_I = \text{span}\{\delta(p - p_\beta)\}\), and defines the collocation/interpolation projection \(\Pi_C : Q \otimes \mathcal{U} \to Q_I \otimes \mathcal{U}\) orthogonal to \(Q_C\) as \(\Pi_C := \tilde{\Pi}\) for \(Q_I = Q_C\). One potential problem is that the norm of the projection \(\|\Pi_C\|\) may become large (much larger than unity), and therefore the collocation/interpolation method may have an issue with stability. In the pseudospectral projection one chooses \(Q_P := \hat{Q}_I = Q_I\), and the projection \(\Pi_P\) so defined is hence orthogonal. Therefore in this case \(\Pi_P := \tilde{\Pi}\) for \(Q_I = Q_P\), and the norm has its minimal value \(\|\Pi_P\| = 1\). In the case of a discrete regression projection, the orthogonality is defined similarly to (10), but with another bilinear form \(\langle \cdot, \cdot \rangle_R\) instead of the inner product \(\langle \cdot, \cdot \rangle_Q\).

So even when choosing \(Q_R := \hat{Q}_I = Q_I\), the projection cannot be orthogonal. Hence in this case \(\Pi_R := \tilde{\Pi}\) for \(Q_I = Q_R\), and \(\|\Pi_R\| \geq 1\). This may again cause problems with stability, although they are typically not as severe as in the case of collocation. For more information on pseudospectral projections and regression or discrete projections, see [12, 15, 43, 44, 11, 49, 7, 31, 6, 50].

In the Galerkin-type methods, it is not the solution \(u^*(p)\), but the residuum \(R(p; u_I(p))\) which is projected. Instead of (13), the approximation \(u_T(p)\) is required.
to satisfy (9). Again the choice $Q_P := \tilde{Q}_I = Q_I$ with the orthogonal projection $\Pi_P$ gives in some sense the “smallest” residuum. The condition (9) is also referred to as Galerkin orthogonality, and it is a basis for efficient and accurate a posteriori error estimators. These in turn can then be used to devise adaptive methods, i.e., choosing the subspace $Q_I$ so that it fits the solution as well as possible. In the approximation of partial differential equations this is already a mature subject, but for parametric and stochastic problems it is in its infancy; see [36] and the references therein.

As already alluded to in the introduction, one may not even choose the subspace $Q_I$ before the computation, but instead build the approximation step by step during the computation. These low-rank tensor approximation methods are again essentially Galerkin methods, and they hold promise for very efficient solution and approximation methods for parametric and stochastic problems; see [24, 10, 45, 18, 46, 23, 19]. It will be important to show that these successive updating methods can also be computed nonintrusively. This work is in preparation.

One may observe that the collocation method appears again through (12) with the projection $H_C$ acting on the residuum, but it is only for special bases $\{\psi_\alpha\}$ in $Q_I$ that satisfy the Kronecker-δ property that we recover the uncoupled equations this way. We refer the reader once more to comparisons of collocation and Galerkin methods in [4, 22] mentioned already in the introduction.

### 3.2. The fixed-point Galerkin equations.

Recall that (2) rather than (1) or (3) will be the starting point for the nonintrusive Galerkin computation. Whatever the starting point, we would still like to achieve the same result. So before continuing, let us show the following.

**Proposition 3.1.** Projecting the fixed-point equation attached to the iteration (2), namely $u_I = u_I + P^{-1}(R(u_I))$, is equivalent to projecting the preconditioned residual $P^{-1}(R(u_I))$; that means for all $\beta \in I$,

\[
G_Q(\phi_\beta(\cdot) P^{-1}(\cdot)(R(\cdot; u_I(\cdot)))) = 0.
\]

Moreover, if the preconditioner $P$ in (5) does not depend on $p$ or $u$, then it is equivalent to projecting the residual $R(u_I)$ from (9); that means for all $\beta \in I$,

\[
G_Q(\phi_\beta(\cdot) R(\cdot; u_I(\cdot))) = 0.
\]

**Proof.** Equation (14) follows simply from linearity of $G_Q$. Furthermore, in the case when $P$ depends on neither $p$ nor $u$, for (15) we have from (14) for any $\beta \in I$,

\[
0 = G_Q(\phi_\beta(\cdot) P^{-1}(R(\cdot; u_I(\cdot)))) = P^{-1} G_Q(\phi_\beta R(u_I)) \iff 0 = G_Q(\phi_\beta R(u_I)),
\]

on noting that for any linear map $L$ on $U$ one has $G_Q(\phi(\cdot) L(\phi(\cdot) v)) = \langle \phi, \phi \rangle_Q L v = L G_Q(\phi(\cdot) \phi(\cdot) v)$, and by observing that $P^{-1}$ is nonsingular. \[\square\]

This means that instead of the residual (3) we may just as well project the iteration (5): with the abbreviation $R^{(k)}(\cdot) := R(\cdot; u^{(k)}(\cdot))$ we have for all $\beta \in I$,

\[
G_Q(\phi_\beta(\cdot) u^{(k+1)}) = G_Q(\phi_\beta(\cdot) u^{(k)} + \Delta u^{(k)}) = G_Q(\phi_\beta(\cdot) u^{(k)} + P^{-1} R^{(k)}).
\]

Expanding $u^{(k)}(p) = \sum_\alpha u^{(k)}_\alpha \psi_\alpha(p)$ in (17), that becomes a coupled iteration equation for the $u_\alpha$:

\[
\forall \beta : G_Q \left( \phi_\beta(\cdot) \sum_\alpha u^{(k+1)}_\alpha \psi_\alpha(\cdot) \right) = G_Q \left( \phi_\beta(\cdot) \left( \sum_\alpha u^{(k)}_\alpha \psi_\alpha(\cdot) + P^{-1} R^{(k)}(\cdot) \right) \right),
\]
which may now be written as

\begin{equation}
\forall \beta: \sum_{\alpha} M_{\beta,\alpha} u_{\alpha}^{(k+1)} = \sum_{\alpha} M_{\beta,\alpha} u_{\alpha}^{(k)} + G_Q(\varphi_\beta P^{-1} R^{(k)}),
\end{equation}

where \( M_{\beta,\alpha} := \langle \varphi_\beta, \psi_\alpha \rangle_Q \). If the coefficients \( u_{\alpha}^{(k)} \in \mathcal{U} \) are arranged columnwise in an \( N \times M \) matrix \( u^{(k)} = [\ldots, u_{\alpha}^{(k)}, \ldots] \in \mathbb{U}^I \), and similarly \( G_Q(P^{-1} R^{(k)}) = [\ldots, G_Q(\varphi_\alpha P^{-1} R^{(k)}), \ldots] \), and the \( M_{\beta,\alpha} \) are viewed as entries of an \( M \times M \) matrix \( M \in \mathbb{R}^{I \times I} \), (19) may be compactly written as

\begin{align}
\begin{aligned}
(20) \quad & u^{(k+1)} M^T = u^{(k)} M^T + G_Q(P^{-1} R^{(k)}) \\
(21) \quad & \text{or} \quad u^{(k+1)} = u^{(k)} + \Delta_Q(u^{(k)}) =: S_Q(u^{(k)}),
\end{aligned}
\end{align}

where we have defined two new functions \( \Delta_Q(u^{(k)}) := [G_Q(P^{-1} R^{(k)})]M^{-T} \) and \( S_Q(u^{(k)}) = u^{(k)} + \Delta_Q(u^{(k)}) \), which will be needed later for the convergence analysis in subsection 3.3.

It is apparent that the computation will be much simplified if the ansatz-functions \( \{\psi_\alpha\}_{\alpha \in \mathcal{I}} \) and the test-functions for the projection \( \{\varphi_\beta\}_{\beta \in \mathcal{I}} \) are chosen biorthogonal, i.e., if one has for all \( \alpha, \beta \in \mathcal{I} \) that \( M_{\beta,\alpha} = \delta_{\beta,\alpha} M_{\alpha,\alpha} \), i.e., \( M = \text{diag}(M_{\alpha,\alpha}) \), as then \( M^{-T} = \text{diag}(1/M_{\alpha,\alpha}) \) is simple to compute and use. Hence,

\begin{equation}
\Delta_Q(u^{(k)}) = G_Q(P^{-1} R^{(k)}) = [\ldots, G_Q(\varphi_\alpha P^{-1} R^{(k)})/M_{\alpha,\alpha}, \ldots].
\end{equation}

For the sake of simplicity of presentation it is assumed from now on that additionally the functions are scaled so that \( M_{\alpha,\alpha} = 1 \), i.e., \( M = I \).

Equation (21) is already a possible way of performing the iteration. The practical, nonintrusive computation of the terms in (21) still has to be considered, but we may formulate the corresponding Algorithm 3.1 and investigate its convergence beforehand. The reader who is interested only in the computational description of the nonintrusive algorithm may jump directly to section 4.

**Algorithm 3.1.** Block-Jacobi iteration of (21).

Start with some initial guess \( u^{(0)} \)

\[ k \leftarrow 0 \]

while no convergence do

\[ \quad \text{Compute } \Delta_Q(u^{(k)}) \text{ according to (22), resp., (21)} \]

\[ \quad u^{(k+1)} \leftarrow u^{(k)} + \Delta_Q(u^{(k)}) \quad [= S_Q(u^{(k)})] \]

\[ \quad k \leftarrow k + 1 \]

end while

Although the underlying iteration (2) in Algorithm 2.1 may be of any kind, e.g., Newton’s method, when one views (21) with regard to the block structure imposed by the \( u = [\ldots, u_\beta, \ldots] \), Algorithm 3.1 is a—maybe nonlinear—block-Jacobi iteration.

**3.3. Convergence of coupled iterations.** Here we want to show that the map \( S_Q \) in (21) satisfies a Lipschitz condition with the same constant as in (6). This will need additional theoretical considerations. For the sake of simplicity we will assume that \( \langle \cdot, \cdot \rangle_Q \) is actually an inner product on the Hilbert space \( Q \subseteq \mathbb{R}^P \) such that \( Q \subseteq Q \). The contraction condition for \( S_Q \) with contraction factor (Lipschitz constant) less
than or equal to \( q^* \) will hold only if the Galerkin projection is orthogonal; i.e., we have to take \( v_{\alpha} = \psi_{\alpha} \), which means \( \hat{Q}_T = Q_T \). Our previous assumption that \( M = I \)—which is now the Gram matrix of the basis \( \{ \psi_{\alpha} \}_{\alpha \in I} \)—now means that this basis is actually orthonormal. In the case when the basis \( \{ \psi_{\alpha} \}_{\alpha \in I} \) is merely orthogonal, we have \( M = \text{diag}(\langle \psi_{\alpha}, \psi_{\alpha} \rangle) \); compare (22).

Parametric elements like \( P \ni p \mapsto u(p) \in \mathcal{U} \) are formally in the Hilbert tensor product space of sums like

\[
\sum_n \phi_n(p) v_n =: \sum_n \phi_n \otimes v_n \in Q \otimes \mathcal{U},
\]

with the inner product of two elementary tensors defined by \( \langle \phi_1 \otimes w_1, \phi_2 \otimes w_2 \rangle_{Q \otimes \mathcal{U}} := \langle \phi_1, \phi_2 \rangle_{\mathcal{Q}} \langle w_1, w_2 \rangle_{\mathcal{U}} \), and then extended by bilinearity. The subspace \( \hat{Q}_T = \text{span} \{ \psi_{\alpha} \}_{\alpha \in I} \subset Q \) is finite-dimensional and hence closed, and therefore leads to the orthogonal direct sum decomposition \( Q = \hat{Q}_T \oplus Q_{\perp}^T \). This implies the orthogonal direct sum decomposition \( Q \otimes \mathcal{U} = (Q \otimes \mathcal{U}) \oplus (Q_{\perp} \otimes \mathcal{U}) \) of the tensor product. Therefore the mapping \( J : \mathcal{U}^T \ni u = [\ldots, u_\alpha, \ldots] \mapsto \sum_{\alpha} \psi_{\alpha}(\cdot) u_\alpha \in Q_T \otimes \mathcal{U} \subseteq Q \otimes \mathcal{U} \), which is by design bijective onto \( Q_T \otimes \mathcal{U} \), may be used to induce a norm and inner product on \( \mathcal{U}^T \) via

\[
\| u \|_{\mathcal{U}^T}^2 := \| Ju \|_{Q \otimes \mathcal{U}}^2 = \| \sum_{\alpha} \psi_{\alpha}(\cdot) u_\alpha \|_{Q \otimes \mathcal{U}}^2 = \sum_{\alpha} \| u_\alpha \|_{\mathcal{U}}^2.
\]

This makes it a unitary map, and hence \( \| J \| = 1 \). When viewed as a mapping into the larger space \( Q \otimes \mathcal{U} \), where it is extended by slight abuse of notation by the inclusion, it remains an isometry with the same norm.

**Lemma 3.2.** The maps \( G_Q : Q \otimes \mathcal{U} \rightarrow \mathcal{U}^T \) and \( J \) are adjoints of each other, \( G_Q^* = J \), and \( G_Q \) is nonexpansive, that is, \( \| G_Q \| = \| G_Q^* \| = 1 \).

**Proof.** For all \( v \in \mathcal{U}^T \) and \( \phi \otimes w \in Q \otimes \mathcal{U} \),

\[
(23) \quad \langle G_Q(\phi \otimes w), v \rangle_{\mathcal{U}^T} = \langle [\ldots, (\psi_{\alpha}, \phi) Q w_\alpha, \ldots], [\ldots, v_\alpha, \ldots] \rangle_{\mathcal{U}^T} = \| v \|_{\mathcal{U}^T}^2 = \langle \phi \otimes w, J v \rangle_{Q \otimes \mathcal{U}},
\]

and hence \( G_Q^* = J \). But \( J \) is an isometry, so that one has \( \| G_Q^* \| = \| J \| = 1 \). As \( \| G_Q \| = \| G_Q^* \| \), we are finished. \( \square \)

With the observation that

\[
(24) \quad G_Q(S(\cdot; u^{(k)}(\cdot), R^{(k)}(\cdot))) = G_Q(u^{(k)}(\cdot) + P^{-1}R^{(k)}(\cdot)) = u^{(k)} + G_Q(P^{-1}R^{(k)}) = u^{(k)} + \Delta_Q(u^{(k)}) = S_Q(u^{(k)}),
\]

the map \( S_Q : \mathcal{U}^T \rightarrow \mathcal{U}^T \) in (21) may be factored in the following way:

\[
(25) \quad S_Q : \mathcal{U}^T \ni S(\cdot; u(\cdot), R(\cdot), R(\cdot, u(\cdot))) = G_Q \circ \tilde{S} \circ J = G_Q \circ \tilde{S} \circ G_Q^*,
\]

(26) \( S_Q = G_Q \circ \tilde{S} \circ J = G_Q \circ \tilde{S} \circ G_Q^* \),

where the symbol \( \circ \) denotes composition of maps, and \( \tilde{S} \) is defined via the solver map \( S \) in (2) by

\[
(27) \quad \tilde{S} : Q \otimes \mathcal{U} \ni u(\cdot) \mapsto S(\cdot; u(\cdot), R(\cdot, u(\cdot))) \in Q \otimes \mathcal{U}.
\]

For this mapping we have the following result.

**Proposition 3.3.** In (27), the map denoted \( \tilde{S} \) has the same Lipschitz constant \( q^* \) as the map \( S \) in (2) (cf. (6)); i.e., \( \tilde{S} \) is a contraction with contraction factor \( q^* < 1 \).

**Proof.** We now use the assumption that the inner product on \( Q \) is given by an integral, \( \langle \phi, \psi \rangle_Q = \int_P \phi(p) \psi(p) \mu(dp) \). In that case \( Q = L_2(P, \mu; \mathbb{R}) \), and the Hilbert
Algorithm 3.1 for the coupled Galerkin system.

The term
\[ \| \tilde{S}(u(\cdot)) - S(v(\cdot)) \|_{L^2(P, \mu; \mathcal{U})}^2 \]
compute the right-hand side of (21), in the form (22). One may observe that
residual for that iteration. Let us recall that the Galerkin projector was defined by

\[ G(u; u) = \int_P S(u(p); R(p; u(p)) - S(v(p), R(p; v(p))))d\mu(dp) \]
\[ \leq (\varrho^*)^2 \int_P ||u(p) - v(p)||_{\mathcal{U}}^2 d\mu(dp) = (\varrho^*)^2 ||u(\cdot) - v(\cdot)||_{L^2(P, \mu; \mathcal{U})}^2, \]
and the proof is concluded by taking square roots.

This immediately leads to the following.

**Corollary 3.4.** The map \( S_\mathcal{Q} \) from (21) is a contraction with contraction factor \( \varrho^* < 1 \) (see (6)):

\[ \forall u, v \in \mathcal{U}^2 : \| S_\mathcal{Q}(u) - S_\mathcal{Q}(v) \|_{\mathcal{U}^2} \leq \varrho^* \| u - v \|_{\mathcal{U}^2}, \]
and hence the Galerkin equations have a unique solution \( u^* \in \mathcal{U}^2 \).

**Proof.** This follows from the decomposition (26), Lemma 3.2, and Proposition 3.3, as \( \| S_\mathcal{Q} \| = \| G_\mathcal{Q} \circ S_\mathcal{Q} \| \leq \| G_\mathcal{Q} \| \| S \| \| G_\mathcal{Q} \| \leq \varrho^* \), and Banach’s contraction mapping theorem.

Now we may state the main result about the convergence of the simple block-Jacobi Algorithm 3.1 for the coupled Galerkin system.

**Theorem 3.5.** As the map \( S_\mathcal{Q} \) from (21) has Lipschitz constant \( \varrho^* < 1 \), and is thus a contraction with the same factor as the solver \( S \) in (2), Algorithm 3.1 converges to the unique solution \( u^* \in \mathcal{U}^2 \) with the same linear speed of convergence as Algorithm 2.1. Additionally, we have the a posteriori error estimate (see (7))

\[ \| u^* - u^{(k+1)} \|_{\mathcal{U}^2} \leq \frac{\varrho^*}{1 - \varrho^*} \| \Delta_Q(u^{(k)}) \|_{\mathcal{U}^2}. \]
The satisfaction of the parametric equation may again be measured by the norm of the residuum \( \| R^{(k)} \|_{\mathcal{Q} \otimes \mathcal{U}} = \| R(\cdot; u^{(k)}) \|_{\mathcal{Q} \otimes \mathcal{U}} \).

**Proof.** Everything simply follows from Corollary 3.4, Banach’s contraction mapping theorem, and the fact that \( R^{(k)}(\cdot) \) is the residuum at iteration \( k \) before any preconditioning or projection.

Observe that the above result holds only for the linear convergence speed. In the case when Algorithm 2.1 has superlinear convergence, this cannot be necessarily matched by Algorithm 3.1. To achieve this, more sophisticated algorithms for the coupled equations are necessary; see, e.g., [41].

4. **Nonintrusive residual.** Here we want to look in more detail at the actual computation of the right-hand side of (21), in the form (22). One may observe that the term \( G_\mathcal{Q}(\varphi_n, P^{-1}R^{(k)}) \) in (22) is the Galerkin projection of the preconditioned residual for that iteration. Let us recall that the Galerkin projector was defined by

\[ G_\mathcal{Q}(\varphi, \psi) = \langle \varphi, \psi \rangle_{\mathcal{Q} \otimes \mathcal{Q}}. \]

4.1. **Analytic computation.** In some cases [40, 37], notably when the preconditioner \( P \) depends on neither \( p \) nor \( u \), or when the operator \( A \) is linear or polynomial in \( u \) and linear in the parameters \( p \), it may be possible to actually represent \( P^{-1}R^{(k)} \), not just in principle, but actually **nonintrusively** through the use of the solver \( S \) in (2) as

\[ P^{-1}R^{(k)}(p) = \sum_n \rho_n(p) v_n = \sum_{n,\beta} \rho_{n,\beta} \psi_{\beta}(p) v_n. \]
For an orthonormal basis the coefficients are simply $\rho_{n,\beta} = \langle \psi_\beta, \rho_n \rangle_G$. The Galerkin projection of this then gives

$$
G_G(\psi_\alpha P^{-1} R^{(k)}) = G_G \left( \psi_\alpha \sum_{n,\beta} \rho_{n,\beta} \psi_\beta v_n \right) = \sum_{n,\beta} \rho_{n,\beta} G_G(\psi_\alpha \psi_\beta v_n) = \sum_{n,\beta} \rho_{n,\beta} \langle \psi_\alpha, \psi_\beta \rangle_G v_n = \sum_n \rho_{n,\alpha} v_n,
$$

using the linearity of $G_G$ and the orthonormality of the basis $\{\psi_\alpha\}_{\alpha \in I}$. This means that for the right-hand side of (21) in the form (22), given the representation (30), each term may be computed through simple linear algebra operations (31). This expression may be directly used in the block-Jacobi algorithm, Algorithm 3.1, for $\Delta_G(u^{(k)})$ in the form (22), and the description of the algorithm is complete. Let us remark finally that if the solver actually returns $S(p; u^{(k)}(p), R^{(k)}(p))$ instead of the increment $P^{-1} R^{(k)}(p)$, then Algorithm 3.1 is easily adapted by computing completely analogously $S_G(u^{(k)})$.

4.2. Numerical integration. The following idea of obtaining a nonintrusive computation of the right-hand side of (21) in the form (22) is more general, but it involves a further approximation, namely numerical integration.

Remembering that it was assumed that the duality pairing on the scalar functions is given by an integral with measure $\mu$,

$$
\langle \varphi, \phi \rangle_G = \int_I \varphi(p) \phi(p) \mu(dp),
$$

we now assume that this integral has some approximate numerical quadrature formula

$$
\int_I \varphi(p) \mu(dp) \approx \sum_{z=1}^Z w_z \varphi(p_z),
$$

where the integrand is evaluated at the quadrature points $p_z$ and the $w_z$ are appropriate weights.

With this approximation the term $G_G(\psi_\beta P^{-1} R^{(k)})$ in (19) becomes practically computable without any further assumptions on the operator $A$, giving

$$
G_G(\psi_\beta P^{-1} R^{(k)}) \approx \Delta_{z,\beta}^G u^{(k)} := \sum_z w_z \psi_\beta(p_z) \Delta u_z^{(k),}\text{ where}
$$

$$
\Delta u_z^{(k)} := P^{-1}(p_z)R(p_z; u^{(k)}(p_z)) = P^{-1}(p_z) \left( f(p_z) - A(p_z; u^{(k)}(p_z)) \right) \text{ or}
$$

$$
= S(p_z; u^{(k)}(p_z), R(p_z; u^{(k)}(p_z))) - u^{(k)}(p_z)
$$

is the preconditioned residuum evaluated at $p_z$, and $u^{(k)}(p_z) = \sum_\alpha u_\alpha^{(k)} \psi_\alpha(p_z)$. This is indeed the only interface needed in the original equation—something which can be easily evaluated nonintrusively as the iteration increment $\Delta u_z^{(k)}$ in (35) in the case when the current state is given as $u^{(k)}(p_z)$ for the parameter value $p_z$. An alternative form is given in (36), which is one iteration of the solver, starting at $u^{(k)}(p_z)$ for the parameter $p_z$. This variant is for the case when the solver actually returns $S(p; u^{(k)}(p), R^{(k)}(p))$ instead of the increment $P^{-1} R^{(k)}(p)$.
4.3. Nonintrusive iteration. The term in (21) in the form of (22) has to be computed nonintrusively. Following subsection 4.2 about numerical integration, one may formulate the approximation of

\[(37) \quad \Delta_Q(u^{(k)}) = [\ldots, G_Q(\varphi_{a} P^{-1} R^{(k)}), \ldots] \approx \Delta_Z(u^{(k)}) = [\ldots, \Delta_{Z,a}u^{(k)}, \ldots]\]

in Algorithm 3.1 from (22) in Algorithm 4.1, using (34) and (35).

Algorithm 4.1. Nonintrusive computation of (22) in the form of (37).

\[
\text{for } \alpha \in I \text{ do} \\
\quad \Delta_{Z,a}u^{(k)} \leftarrow 0 \\
\text{end for} \\
\triangleright \text{ %comment: the loop over integration points}%
\text{for } z \leftarrow 1, \ldots, Z \text{ do} \\
\quad \text{Compute } \Delta u_z^{(k)} \text{ from (35)} \\
\quad r_z \leftarrow w_z \Delta u_z^{(k)} \\
\quad \text{for } \alpha \in I \text{ do} \\
\qquad \Delta_{Z,a}u^{(k)} \leftarrow \Delta_{Z,a}u^{(k)} + \psi_{a}(p_z) r_z \\
\text{end for} \\
\text{end for}
\]

The result of this algorithm is \(\Delta_Z(u^{(k)})\) in (37), the approximation of \(\Delta_Q(u^{(k)})\) by numerical integration. With Algorithm 4.1 it is now possible to formulate a nonintrusive version of Algorithm 3.1, the block-Jacobi iteration, in Algorithm 4.2.

The sequence generated by Algorithm 4.2 has been labeled with a tilde \(\tilde{u}^{(k)}\) to distinguish it from the exact sequence \(\{u^{(k)}\}_k\) generated by Algorithm 3.1. The questions arise as to how well the original sequence \(\{u^{(k)}\}_k\) is approximated by the one produced nonintrusively by numerical integration \(\{\tilde{u}^{(k)}\}_k\), and what its convergence behavior is.

Algorithm 4.2. Nonintrusive block-Jacobi iteration of (21).

\[
\text{Start with some initial guess } \tilde{u}^{(0)} = [\ldots, \tilde{u}_{a}^{(0)}, \ldots] \\
\text{for } k \leftarrow 0 \text{ do} \\
\quad \text{while no convergence do} \\
\qquad \text{Compute } \Delta_Z(\tilde{u}^{(k)}) = [\ldots, \Delta_{Z,a}\tilde{u}^{(k)}, \ldots] \text{ according to Algorithm 4.1} \\
\qquad \text{ } \quad \tilde{u}^{(k+1)} \leftarrow \tilde{u}^{(k)} + \Delta_Z(\tilde{u}^{(k)}) \\
\qquad \text{ } \quad k \leftarrow k + 1 \\
\text{end while}
\]

To that effect we partially cite and conclude from Theorem 4.1 in [42].

**Theorem 4.1.** Assume that the numerical integration in Algorithm 4.1 is performed such that \(\|G_Q(\psi_{a} P^{-1} R(\cdot; \tilde{u}^{(k)})) - \Delta_{Z,a}\tilde{u}^{(k)}\|_U \leq \varepsilon/\sqrt{M}\); then the error in (37) is estimated by

\[(38) \quad \|\Delta_Q(\tilde{u}^{(k)}) - \Delta_Z(\tilde{u}^{(k)})\|_U \leq \varepsilon,
\]

and we have the following a posteriori error estimate for the iterates:

\[(39) \quad \|u^* - \tilde{u}^{(k+1)}\|_U \leq \frac{\varrho^*}{1 - \varrho^*}\|\Delta_Z(\tilde{u}^{(k)})\|_U + \frac{\varepsilon}{1 - \varrho^*}.
\]
In addition, we have that

\[
\limsup_{k \to \infty} \| \mathbf{u}^* - \tilde{\mathbf{u}}^{(k)} \|_{\mathbf{U}^*} \leq \frac{\varepsilon}{1 - \rho^*}.
\]

The satisfaction of the parametric equation may be measured by \(\| R(\cdot; \tilde{\mathbf{u}}^{(k)}) \|_{Q \otimes \mathbf{U}}\).

Proof. Equation (38) is a simple consequence of the assumption by squaring and summing \(M = |\mathcal{I}|\) terms of size less than \(\varepsilon/\sqrt{M}\), and then taking the square root. Everything else are then statements of Theorem 4.1 in [42]. \(\square\)

Equation (39) shows that the modified sequence \(\{\tilde{\mathbf{u}}^{(k)}\}_k\) will not necessarily converge to \(\mathbf{u}^*\), even if \(\Delta_Z(\tilde{\mathbf{u}}^{(k)}) \to 0\) as \(k \to \infty\), but (40) shows that it clusters around \(\mathbf{u}^*\) in a small neighborhood.

4.4. Computational effort and possible improvements. To assess the effort involved in a computational procedure and hence its efficiency is always difficult, partly because it is not always clear how to measure computational effort. Here we take the view that the effort is only counted in solver calls, i.e., invocations of \(p\) involved in a computational procedure and hence its efficiency is always difficult.

The approximation in Algorithm 4.3 to (41) is computed very similarly to that in Algorithm 4.1.

The main contender for the Galerkin procedure outlined so far is the so-called pseudospectral or discrete projection, also called a regression. This can be described very quickly. With \(\{\psi_\alpha\}_{\alpha \in \mathcal{I}}\) orthonormal, the coefficients in the projection \(u_\mathcal{I} = \sum_{\alpha \in \mathcal{I}} u^* \psi_\alpha\) can be simply computed by inner products:

\[
u^\alpha = \langle \psi_\alpha, u^* \rangle_Q = \int_p \psi_\alpha(p) u^*(p) \mu(dp) \approx \sum_{z=1}^Z w_z \psi(p_z) u^*(p_z).
\]

One may be reminded that this—being the orthogonal projection onto the subspace \(Q_\mathcal{I} \subseteq Q\)—has the smallest error to \(u^*(p)\) in the norm \(\| \cdot \|_Q\), but it does not at all take into account the parametric equation. The Galerkin projection, on the other hand, will produce an approximation which is optimal in minimizing the residuum. The approximation in Algorithm 4.3 to (41) is computed very similarly to that in Algorithm 4.1.

One may observe that the basic Algorithm 2.1 for (2) for a fixed parameter value converges with a contraction factor not worse than \(\rho^*\). Say that an iteration with contraction factor of \(\rho^*\) needs \(L\) iterations to converge to the desired accuracy, and at each iteration one solver call is needed. Hence the corresponding step in the projection algorithm, Algorithm 4.3, also needs at most \(L\) iterations for each parameter value \(p_z\). This is executed in the loop over the integration points, i.e., \(Z\) times. Therefore Algorithm 4.3 needs at most \(L \times Z\) solver calls.

The block-Jacobi variant of the nonintrusive coupled Galerkin system in Algorithm 4.2 needs one solver call for each of the \(Z\) integration points per iteration, i.e., a total of \(Z\) solver calls per iteration. As shown in Corollary 3.4, it converges also
Algorithm 4.3. Discrete projection according to (41).

\begin{verbatim}
for \( \alpha \in I \) do
    \( u^\alpha \leftarrow 0 \)
end for
\end{verbatim}

\[ \triangleright \text{comment: the loop over integration points}\]

\begin{verbatim}
for \( z \leftarrow 1, \ldots, Z \) do
    Compute \( u(p_z) \) according to Algorithm 2.1.
    \( r_z \leftarrow w_z u(p_z) \)
    for \( \alpha \in I \) do
        \( u^\alpha \leftarrow u^\alpha + \psi_\alpha(p_z) r_z \)
    end for
end for
\end{verbatim}

with a contraction factor not worse than \( \varrho^* \); hence it needs at most \( L \) global block-Jacobi iterations. This again gives a total of \( L \times Z \) solver calls.

We see that if the measure of effort is solver calls, then the discrete projection and the block-Jacobi iteration of the Galerkin system need the same effort for comparable accuracy; something that is borne out also in the numerical example in section 5.

When looking at the other computations apart from the count of solver calls, in both algorithms integrals have to be approximated by quadrature formulas. In the discrete projection this happens only once, whereas in the block-Jacobi this is done in every global iteration. But as explained in the beginning of this section, in the situation where an emulator is at all interesting, these few additional operations of linear algebra are negligible when compared to the cost of a solver call.

In the case when the iteration in (2) is superlinearly convergent, e.g., if it is Newton’s method, then this cannot be matched by the block-Jacobi method; it will usually only have linear convergence. But even Newton’s method [35] can be emulated on the global Galerkin system, where the action of the inverse of the derivative on a vector is approximated by finite differences realized through nonintrusive solver calls. This last procedure is even able to maintain quadratic convergence. We refer the reader to [35, 41] for further details.

Block-Jacobi is probably the simplest method for coupled systems; however, it can be considerably accelerated [35, 41]: this ranges from the simple Aitken acceleration over block Gauss–Seidel to quasi-Newton methods. In the case when the iterations from (2) converge only linearly, these extensions can then produce an advantage for the Galerkin solution and may need considerably fewer than \( L \) iterations, as “convergence information” is shared for different values of \( p \) or \( \alpha \)—something which will not happen in the decoupled discrete projection.

A considerable saving of work in the Galerkin procedure is also possible with sparse or low-rank approximations. They come about when viewing the solution as tensors, which may be used computationally in low-rank representations/approximations; see, for example, [42, 23]. Although such extensions are possible, they are beyond the scope of the present work.

5. Numerical example. Here we want to show the procedures discussed on a tiny example which nonetheless is representative of parametric problems. It is so simple that it may be programmed with a few lines of code. This computational example derives from a little electrical resistor network with a global nonlinearity. The particular resistor network we use is shown in Figure 1.
Kirchhoff’s and Ohm’s laws lead to the following linear relation between voltages \( u \) and currents \( j \) fed into the nodes, where the numbering of the nodes corresponds to the equations—node 6 is grounded \( (u_6 = 0) \) and so needs no equation; hence \( u \in U = \mathbb{R}^5, K \in \mathbb{R}^{5 \times 5} \):

\[
K u = j,
\]

with

\[
K = \frac{1}{R} \begin{bmatrix}
3 & -1 & -1 & 0 & -1 \\
-1 & 3 & -1 & -1 & 0 \\
-1 & -1 & 4 & -1 & -1 \\
0 & -1 & -1 & 3 & -1 \\
-1 & 0 & -1 & -1 & 4
\end{bmatrix},
\]

This simple form of the matrix stems from the fact that we have chosen all resistors to have the same resistance, which in this case is \( R = 1/100 \).

To make this toy system nonlinear, we add a global cubic nonlinearity with an uncertain coefficient \( \lambda_1(p_1)(u^T u) u \), where \( \lambda_1 \) is a function of \( p_1 \) to be specified on a per example basis later to allow different variants to be computed. This term has no particular physical model attached to it and is just a simple nonlinearity. If one wishes, one may interpret it as an additional conductivity between all nodes and the grounding point node 6, where the conductivity increases with the total dissipated power, which in turn is proportional to \( (u^T u) \).

We also make the feed-in current \( f(p) = \lambda_2(p_2)f_0 \) uncertain, where \( \lambda_2 \) is a function of \( p_2 \), to be specified on a per example basis later, so that eventually we are left having to solve this system for \( u \) (this is a concrete example of (1)):  

\[
A(p; u) := (K u + \lambda_1(p_1)(u^T u) u) = \lambda_2(p_2)f_0 =: f(p),
\]

\[
f_0 := [1, 0, 0, 0, 0]^T,
\]

where the random parameters \( p = (p_1, p_2) \) are assumed uniformly and independently distributed in \([-1, 1]\), and therefore we have for the residuum (compare (3))

\[
R(p; u) = \lambda_2(p_2)f_0 - (K u + \lambda_1(p_1)(u^T u) u),
\]
and for the preconditioner we take simply $P = K = D_u A(p; 0)$.

The system can be solved in an iterative way as formulated in (2) with, effectively,

$$(47) \quad u^{(k+1)} = S(p; u^{(k)}, R(p; u^{(k)})) = u^{(k)} + P^{-1} R(p; u^{(k)}) \quad = K^{-1} \left( \lambda_2(p_2) f_0 - \lambda_1(p_1) \left( (u^{(k)})^T u^{(k)} \right) u^{(k)} \right).$$

The simple iteration (47), which is indeed a linearly convergent modified Newton method, converges quite well for the chosen parameters.

For the ansatz-functions we take tensor products of Legendre polynomials, as they are orthogonal for the uniform measure; i.e., we take $\psi_\alpha(p) = L_\alpha(p) = L_\alpha(p)/\|L_\alpha\|$, the multivariate normalized Legendre polynomial, and $L_\alpha(p) = \prod_{i=1}^2 \ell_\alpha(p_i)$, where the $\ell_1$ are the normal univariate Legendre polynomials, the norm is given by $\|L_\alpha\| = 4(2\alpha_1 + 1)^{-1}(2\alpha_2 + 1)^{-1}$, and $\alpha \in (N_0)^2$:

$$(48) \quad u(p) \approx \sum_{|\alpha| \leq m} u_\alpha \hat{L}_\alpha(p) =: u_T(p), \quad \text{with}$$

$$u_\alpha = [u_{\alpha,1}, \ldots, u_{\alpha,5}]^T \in \mathcal{U} = \mathbb{R}^5, \quad \text{and}$$

$$\mathcal{I} = \{ \alpha = (\alpha_1, \alpha_2) : |\alpha| = \alpha_1 + \alpha_2 \leq m \} \subset (N_0)^2, \quad m \in \mathbb{N};$$

hence for different $m \in \mathbb{N}$ we will have different approximation orders by polynomials of total degree $m$.

For the purpose of comparison we use two approaches to determine the coefficients $u_\alpha$ in (48): these are the Galerkin approach according to Algorithm 4.2 with numerically integrated residuum according to Algorithm 4.1, and the discrete projection with numerical integration according to Algorithm 4.3, both with the same integration rule. We choose here—as we are only in two dimensions—a tensor-product Gauss–Legendre quadrature. The quadrature order was always taken so that products of test- and ansatz-functions $\psi_\alpha \psi_\beta$ were integrated exactly for the chosen total polynomial degree $m$ in (48).

Four different test cases of the simple example in (44) will be shown, with different choices for the coefficient functions. Table 1 shows the test cases. The first row shows the dependence of the nonlinear term on the parameter $p_1$, while the second row shows the dependence of the right-hand side $f$ on the parameter $p_2$. For each case, we compute the average or mean value of the solution $\bar{u} = \int_{[-1,1]^2} u(p) \, dp$, and from this the norm of the mean $\| \bar{u} \|$, as well as the fluctuating part $u(p) = u(p) - \bar{u}$ and its norm $\| u \|$. The ratio of the two norms is the \textit{coefficient of variation}, c.o.v. = $\| u \|/ \| \bar{u} \|$, which is shown in Table 1 in the third row.

The first three test cases have a simple affine parametric dependence. The first test case has a moderate level of nonlinearity, and only very little randomness, as may be gleaned from the small c.o.v. The second test case has about the same level of nonlinearity, but much more randomness, as evidenced by a c.o.v. which is three orders of magnitude larger. The third test case has the strongest nonlinearity but a moderate c.o.v. Both the second and third test cases are on the borderline for the convergence of the simple iteration from (47), which is the basic “building block” here. This means that for a stronger nonlinearity or for a higher c.o.v. the iteration (47) will not converge at all sampling points, and hence the projection (collocation)
method would no longer work. The fourth and last test case has again a moderate c.o.v. and strong nonlinearity, but the parametric dependence is no longer affine.

Three error measures were computed to compare the results, namely,

\begin{align}
\epsilon(L_2(u)) & := \| u(p) - u_T(p) \|_{L_2} = \left( \int_{[-1,1]^2} \| u(p) - u_T(p) \|^2_{\mathbb{R}^5} \, dp \right)^{1/2}, \\
\epsilon(L_1(u)) & := \| u(p) - u_T(p) \|_{L_1} = \int_{[-1,1]^2} \| u(p) - u_T(p) \|_{\mathbb{R}^5} \, dp, \\
\epsilon(L_2(R_u)) & := \| R(p; u_T(p)) \|_{L_2} = \left( \int_{[-1,1]^2} \| R(p; u_T(p)) \|^2_{\mathbb{R}^5} \, dp \right)^{1/2},
\end{align}

where \( u(p) \) is the exact solution, \( u_T(p) \) is the approximation as in (48), and \( R(p; u_T(p)) \) is the residuum from (46). The error measure in (49) is the \( L_2 \) norm in \( Q \otimes U \cong L_2([-1,1]^2; \mathbb{R}^5) \) of the error, which is also labeled as root-mean-squared-error (RMSE). The second error measure in (50) is the \( L_1([-1,1]^2; \mathbb{R}^5) \) norm of the error, and the third in (51) is the \( L_2 \) norm of the residuum. One may recall that the projection method computes the polynomial approximation, which minimizes the criterion (49), whereas the Galerkin method computes the polynomial approximation, which minimizes the criterion (51).

The above integrals for the error measures were computed with an \( N = 10^7 \) sample Monte Carlo integration to high accuracy by setting the convergence criterion in Algorithm 2.1 for the exact solution \( u(p) \) to 10 times the machine tolerance. The number of Monte Carlo samples was chosen so that we have 99% confidence that the integration error in the computed error measures is less than 1%.

In the computation of the approximation \( u_T(p) \) as in (48) by the two methods, projection and nonintrusive Galerkin, it does not make much sense to iterate too far when the polynomial order is low, as this does not improve the overall accuracy. This is shown in Figure 2 for the first case, where the overall RMSE in (49) is displayed for each polynomial order in dependence of the iteration tolerance of the solver: on the left for the projection method and on the right for the nonintrusive Galerkin method. The criterion of convergence for the iterative solvers was that the increment of \( u \) or \( u_{\alpha} \) be smaller than \( \epsilon_{\text{tol}} \). In the following comparison we have stopped the iterations at the point where the curves level off. We point out that, as the iterations are different for the two methods and the convergence tolerance on the horizontal axis is not measured in the same way, this kind of termination criterion will tend to favor the projection method, as this method is the best one for minimizing the RMSE (49).

The two approaches were carried out to compute the coefficients \( u_{\alpha} \) in (48). The results for the first test case from Table 1 are shown in Table 2. The achieved
accuracies of the approximation are very good and the errors are very small, with no discernible difference for both methods, no matter which error measure is taken. One may also observe that the count for the number of solver calls is about the same, with the projection method a bit more favorable as predicted. This first test case had low c.o.v. and moderate nonlinearity.

In the second test case, the results of which are shown in Table 3, the nonlinearity is about the same, but the c.o.v. is three orders of magnitude higher. As a result, the error measures are about two to four orders of magnitude larger than in the previous case, but still practically about the same for both methods. Interestingly, the iteration counts are lower than for the first case in Table 2. This seems at first sight counterintuitive. But due to the much higher c.o.v., the best polynomial approximation for a given degree is significantly less accurate, and hence the leveling off as in Figure 2 occurs much earlier, and hence the iterations are not needed to such high accuracy, resulting in fewer solver calls.

The results for the third test case are shown in Table 4. Again here, as in the previous test case in Table 3, the accuracies for the two methods are practically indistinguishable. The c.o.v. is now only moderate, but the nonlinearity is very strong. Thus the achieved accuracy is better than in the second case but not as good as in the first, and the iteration count is significantly higher due to the nonlinearity. The results for the fourth test case are shown in Table 5. Here the parametric dependence is very different, the c.o.v. is only moderate, but the nonlinearity is still high. Due to the different parametric dependence, the error norms for the polynomial approximations are worst for all test cases here. But apparently now many iterations, and hence solver calls, are necessary to achieve convergence even to a moderate level. Here there
is also a discernible difference between the two methods, with the Galerkin method
giving slightly more accurate results than the projection method, though it is not so
big as to be really significant.

The results essentially confirm the theoretical analysis in section 3; for the same
accuracy both approaches need about the same number of solver calls; i.e., the simple
block-Jacobi iteration of the Galerkin system converges at essentially the same speed
as the original iteration. We remark that it is not easy to compare the methods for
both criteria, namely, number of solver calls and error measures, on a level playing
field. Of course one could have run the iterations much further, squeezing the error
measures as much as possible. Here one would expect the projection method to give
the best results for the RMSE $\varepsilon(L_2(u))$, and the Galerkin method the best results
for $\varepsilon(L_2(R_u))$, as this is what the methods minimize. But then the iteration counts
would be hardly comparable. Or one could have chosen a fixed number of solver calls
for each computation as the “affordable expense” and then compared the achieved
accuracy. The problem here is that the convergence behavior is not always regular
for all error measures, so that this comparison may be sort of unfair in other ways.

Instead we have chosen to run the iteration only to the point where the RMSE no
longer improves, implicitly favoring the projection method slightly in terms of solver
calls. This is what can be seen in all the test cases. The main result from the tests is
to confirm that it is possible to apply the Galerkin method nonintrusively, and that
even for the simple block-Jacobi iteration the number of solver calls behaves similarly
to the projection method. The possible advantages of a Galerkin solution have already
been alluded to and will once more be picked up in the conclusion.
Table 5
Case 4: # of solver calls and error norms; P = projection, G = Galerkin.

<table>
<thead>
<tr>
<th>Order</th>
<th>Solver calls</th>
<th>$\epsilon(L_2(u))$</th>
<th>$\epsilon(L_1(u))$</th>
<th>$\epsilon(L_2(R_u))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>P</td>
<td>G</td>
<td>P</td>
<td>G</td>
</tr>
<tr>
<td>2</td>
<td>38</td>
<td>27</td>
<td>1.3e-1</td>
<td>1.2e-1</td>
</tr>
<tr>
<td>3</td>
<td>61</td>
<td>80</td>
<td>7.5e-2</td>
<td>7.4e-2</td>
</tr>
<tr>
<td>4</td>
<td>147</td>
<td>250</td>
<td>4.2e-2</td>
<td>4.0e-2</td>
</tr>
<tr>
<td>5</td>
<td>218</td>
<td>324</td>
<td>3.4e-2</td>
<td>2.8e-2</td>
</tr>
</tbody>
</table>

6. Conclusion. After reviewing the literature on numerical methods for parametric equations, with a special emphasis on the subclass of stochastic equations, we have introduced a general methodology to formulate numerical methods relying on functional or spectral approximations. We have shown that the Galerkin orthogonality conditions for the residuum and the iteration equation are equivalent under certain conditions, and that the simplest iterative scheme for the coupled Galerkin system, the block-Jacobi method, converges essentially at the same speed as the original solver for a single parameter value.

In subsections 4.2 and 4.3 for this “plain vanilla” Galerkin formulation, we have shown how to approximate the preconditioned residuum in the Galerkin equation through numerical integration, and have shown the effects of this on the iteration sequence. One may note that we have not said anything about the numerical integration procedure, except to require that it be performed with some minimum accuracy. But these are integrals in potentially high-dimensional spaces, and therefore not trivial. We would like to point out that any acceleration techniques developed for high-dimensional integrals would apply directly and be of tremendous benefit here.

Then these explicit nonintrusive Galerkin algorithms were compared on several test cases of a simple, easy-to-understand example. This showed that the theoretical analysis was validated with these computations and that even in the simplest case of block-Jacobi the Galerkin formulation is competitive with collocation and projection. We have also shown how it performs on cases where the variance is high, where there is considerable loss of accuracy. The method is still stable, which in our view is due to the inherent stability of projections.

This demonstration of “plain vanilla” Galerkin computation, together with the fact that it can be done nonintrusively, was the main purpose of this paper. This is important not because of this particular procedure, which is the simplest nonintrusive Galerkin formulation, but rather in view of the possibilities which this opens. We have alluded to these extensions already in section 1, and in particular in subsection 3.1 for the more theory-based aspects, and in subsection 4.4 regarding the possibilities of accelerating the coupled Galerkin solution, something that is not possible for the decoupled collocation and projection approach. In short, this means that on the theoretical side, due to Galerkin orthogonality it becomes possible to develop a posteriori error estimators, and hence adaptive procedures. On the computational side, this opens possibilities of accelerating global iteration schemes, which turn the fact of coupled equations into an advantage. This should certainly be combined with the adaptive approaches mentioned in subsection 3.1. It also points to the possibility of
nonintrusive variants of schemes which slowly build the solution through successively more accurate low-rank tensor approximation, such as successive rank-one updates and proper generalized decomposition (PGD).

REFERENCES


© 2014 Loïc Giraldi, Alexander Litvinenko, Dishi Liu, Hermann G. Matthies, Anthony Nouy


