SPIRRID – tool for estimation of statistical characteristics of functions with multivariate random inputs

VÁCLAV SADÍLEK and
MIROSLAV VOŘECHOVSKÝ
Brno University of Technology
Faculty of Civil Engineering
Institute of Structural Mechanics
Veveří 331/95, 602 00 Brno
CZECH REPUBLIC
sadilek.v@fce.vutbr.cz

ROSTISLAV CHUDOBA and
ROSTISLAV RYPL
RWTH Aachen University
Faculty of Civil Engineering
Institute of Structural Concrete
Mies-van-der-Rohe-Strae 1, D-52074 Aachen
GERMANY
rostislav.chudoba@rwth-aachen.de

Abstract: This paper is a promotion of a recently published journal paper [1]. The paper presents application of known mathematical methods and computational techniques for the numerical integration in the computational tool SPIRRID (Python package) for efficient and flexible evaluation of the statistical characteristics of functions with multivariate random inputs developed in high-level programming language Python. The paper describes mathematical formulation of the estimation of the statistical characteristics, design of an object oriented architecture of the package and results of performance studies. The code of this computational tool including examples and the results of performance studies is available via the free repository [2].

Key–Words: Python, Enthought Traits, NumPy, SciPy, C, statistical characteristics, multidimensional integration

1 Introduction

SPIRRID is an open source project with object oriented architecture for estimation of the statistical characteristics of functions with multivariate random inputs. This computational tool was developed in high-level programming language Python. High-level languages for scientific computing offer application programmers a convenient and efficient tool for the formulation and implementation of mathematical models. SPIRRID package uses two rich numerical libraries numpy and scipy. These libraries embody algorithms and methods developed over the past decades in the compiled languages FORTRAN and C++. As further documented in [3], the high flexibility of the scripting language is not necessarily accompanied by a lower efficiency with respect to the compiled code. Indeed, when implementing mathematical expressions in vectorized form using compact array representation [4], the tradeoff between flexibility and efficiency is reduced to an acceptable level [5].

The presentation and explanation of the package SPIRRID is performed using an illustrative example that features a simple response function depicted in Fig. 1 (left):

$$q(\varepsilon;\lambda,\xi) = \lambda\,\varepsilon\cdot H\left(\xi-\varepsilon\right). \tag{1}$$

The generalized implementation called SPIRRID is applicable for the statistical characterization of an arbitrary function with independent random parameters. It can be used for statistical analyses of other engineering problems, such as e.g. [6, 7]. The results of performance studies and figures contained in [1] and this paper are calculated using function (1), too. This function defines the stress for a given positive control strain ε of a linear elastic, brittle fiber loaded in tension with the stiffness parameter λ and breaking strain ξ. The symbol $H(x)$ represents the Heaviside function returning zero for $x<0$ and one for $x\geq 0$. We chose a function containing discontinuity in order to study the ability of the integration algorithm to reproduce the smoothness of the average response of an ensemble with a constituent response governed by a non-smooth function. If many fibers act in parallel, as is the case in a crack bridge in a composite, they exhibit imperfections. Then, material parameters λ and ξ are considered independent, identically distributed random parameters, see Fig. 1, left. Random realizations of the single fiber's response are displayed in Fig. 1, right. The mean stress-strain behavior of a fiber within a bundle can be obtained as

$$\mu_q(\varepsilon) = \sum_{\Theta_\lambda}\sum_{\Theta_\xi} \underbrace{q\left(\varepsilon;\lambda,\xi\right)}_{Q}\,\underbrace{g_\lambda g_\xi\,\Delta\theta_\lambda\Delta\theta_\xi}_{\Delta G} \tag{2}$$

where g_λ and g_ξ are the densities of random parameters λ and ξ, respectively. The result of this expression is plotted in Fig. 1 (right, black curve). It demonstrates

ISBN: 978-1-61804-117-3

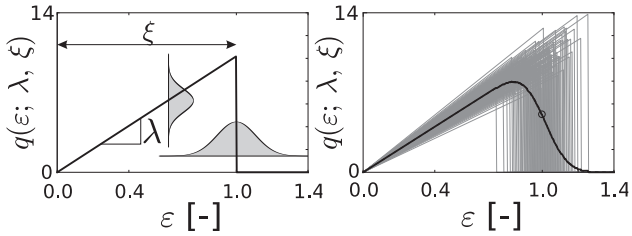


Figure 1: Left: elementary response described by a function with two random parameters; Right: sample of random responses (gray) and the calculated mean response (black).

that the average response of the filament is nonlinear. The described statistical integral exhibits the structure of a strain-based fiber bundle model describing the behavior of yarns and composite materials [8]. The authors have used the procedure for modeling the tensile tests of multi-filament yarns [9].

2 Estimation of statistical characteristics of a function with independent random variables

The goal is to estimate the statistical moments of a random variable Q given as a function of a control variable ε and of random variables $\theta_1 \dots \theta_m$:

$$Q = q\left(\varepsilon; \theta_1, \dots, \theta_m\right) \tag{3}$$

The k-th raw statistical moment of such a variable is given as

$$\mu_k(\varepsilon) = \int_{\boldsymbol{\Theta}} \left[q(\varepsilon; \boldsymbol{\theta})\right]^k g(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta}. \tag{4}$$

Since only independent random variables are considered here, the joint probability density function $g(\boldsymbol{\theta})$ (PDF) of the random vector $\boldsymbol{\theta}$ can be replaced with the product of univariate marginal densities

$$\begin{aligned} \mu_k(\varepsilon) = & \int_{\Theta_1} \dots \int_{\Theta_m} \left[q(\varepsilon; \boldsymbol{\theta})\right]^k \\ & \cdot g_1(\theta_1) \dots g_m(\theta_m) \, \mathrm{d}\theta_1 \dots \mathrm{d}\theta_m. \end{aligned} \tag{5}$$

The integration is to be performed numerically as a summation of discrete values distributed over the discretized random domain

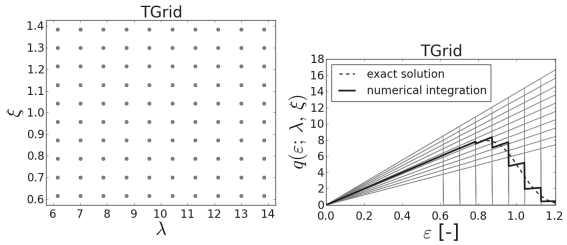
$$\begin{aligned} \mu_k(\varepsilon) = & \sum_{\Theta_1} \dots \sum_{\Theta_m} \underbrace{\left[q\left(\varepsilon; \theta_1 \dots \theta_m\right)\right]^k}_{Q^k} \\ & \cdot \underbrace{\Delta G_1(\theta_1) \dots \Delta G_m(\theta_m)}_{\Delta G}, \end{aligned} \tag{6}$$

where $\Delta G_i(\theta_i)$ denote the weighting factors that depend on the particular type of sampling as specified below. The distribution of the integration points Θ_m within the random domain can be expressed as

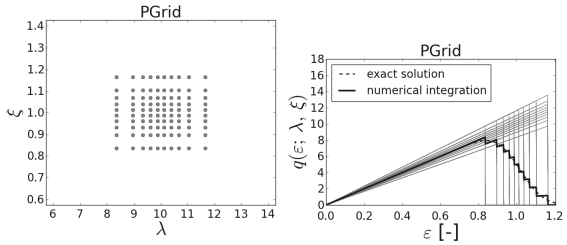
$$\Theta_i = [\theta_{ij}, j \in 1 \dots n_i], \; i \in 1 \dots m, \tag{7}$$

where n_i is the number of discretization points for the i-th variable and j counts the disretization point along a variable. There are many ways to cover the random domain by sampling points. In order to demonstrate the expressive power of the language and to discuss the computational efficiency of the possible implementation techniques we shall implement the integral (5) in four different ways:

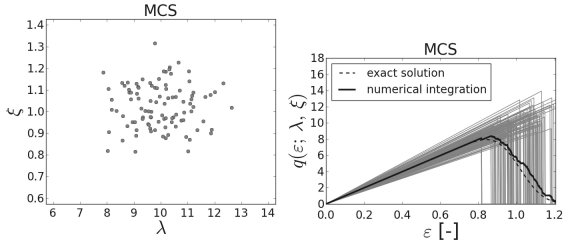
- Equidistant grid of random variables (TGrid)



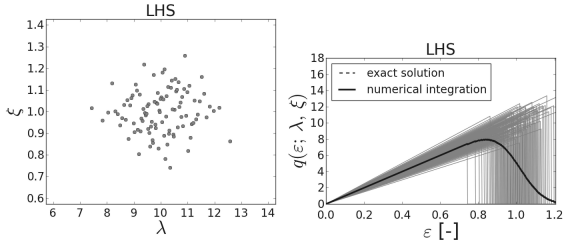
- Non-equidistant grid of random variables generated through an equidistant grid of sampling probabilities (PGrid)



- Crude Monte Carlo Sampling (MCS)



- Latin Hypercube Sampling (LHS)



The detailed description of these four disretization methods can be found in the journal paper [1].

3 Implementation of the SPIRRID class using traits

Let us shortly explain the approach to SPIRRID class implementation. Figure 2 shows the structure of the code in the form of a UML class diagram [10]. The problem is decomposed into three parts:

randomization: representing the specification of the random problem (definition of the function and its control and random parameters).

sampling: delivering the sampling data for the given randomization.

code generation: providing the instantiated execution method (either vectorized code or a compiled C code).

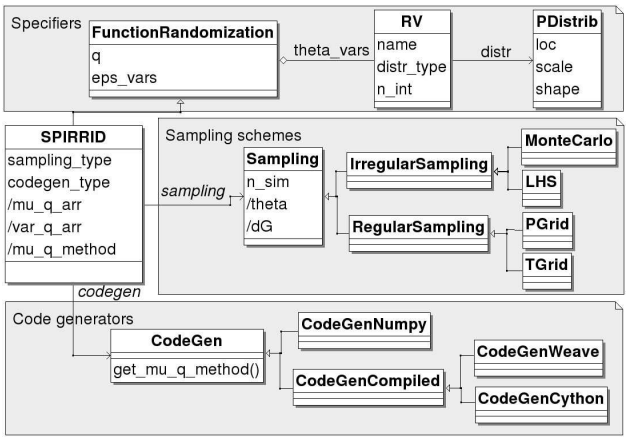


Figure 2: UML class diagram with a general representation of the integration algorithm and extensible components for sampling schemes and implementations. Attribute names preceded by slash indicate derived output values implemented as property traits.

Our Python implementation of the class diagram uses the traits [11, 12] package provided within the *Enthought Tool Suite* to glue these parts together. The package traits introduces an extended attribute definition into Python classes by including specification of the attribute's type and of its dynamic behavior (Initialization, Validation, Delegation, Notification and Visualization). Such an extended attribute specification has been used to incorporate the Model-View-Controller design pattern into the language and provide automatic object visualization and dynamic control of state changes with minimum programming effort. This semantically rich language support allows

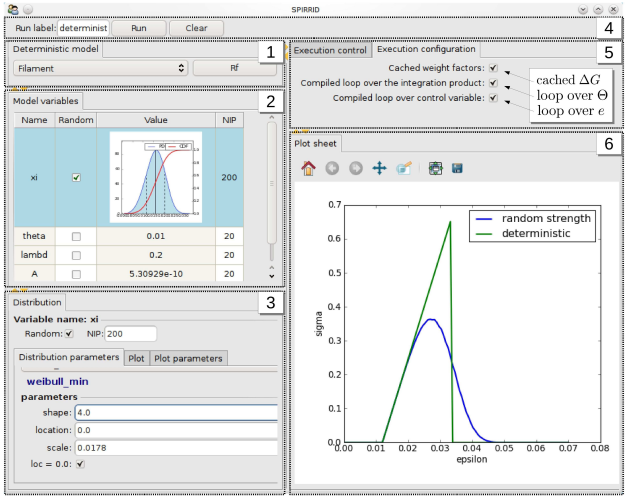


Figure 3: User interface view of the SPIRRID class.

developers to benefit from automatic generation of the user-interface (Fig. 3) directly from the class view specification, persistence of objects, declaration of state dependencies and effective support for data visualization.

4 Computational efficiency

The mean response of the two-parametric response function given in Eq. (1) with both parameters considered random and normally distributed can be solved analytically:

$$\mu_q^{\text{exact}}(\varepsilon) = \frac{\mu_\lambda \varepsilon}{2}\left(1 - \operatorname{erf}\left(\frac{\sqrt{2}(\varepsilon - \mu_\xi)}{2\sigma_\xi}\right)\right) \quad (8)$$

where $\operatorname{erf}(x)$ denotes the Gauss error function defined as $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t}\,\mathrm{d}t$. The convergence of the sampling schemes used for the numerical estimation of the mean has been studied depending on the number of sampling points and on executional time. Two errors have been formulated, both local and global. The local error measure has been defined as the relative maximum deviation along the control variable ε discretized using n_ε points:

$$\mathrm{e}_{\max} = \frac{\max_i \left|\mu_q(\varepsilon_i) - \mu_q^{\text{exact}}(\varepsilon_i)\right|}{\max_i \left[\mu_q^{\text{exact}}(\varepsilon_i)\right]}, \quad (9)$$

where $i = 1 \ldots n_\varepsilon$. The global error measure has been introduced as the relative root mean square error within the range of the control variable ε:

$$\mathrm{e}_{\text{rms}} = \frac{\sqrt{\frac{1}{n_\varepsilon}\sum_i \left(\mu_q(\varepsilon_i) - \mu_q(\varepsilon_i)^{\text{exact}}\right)^2}}{\max_i \left[\mu_q^{\text{exact}}(\varepsilon_i)\right]}, \quad (10)$$

ISBN: 978-1-61804-117-3

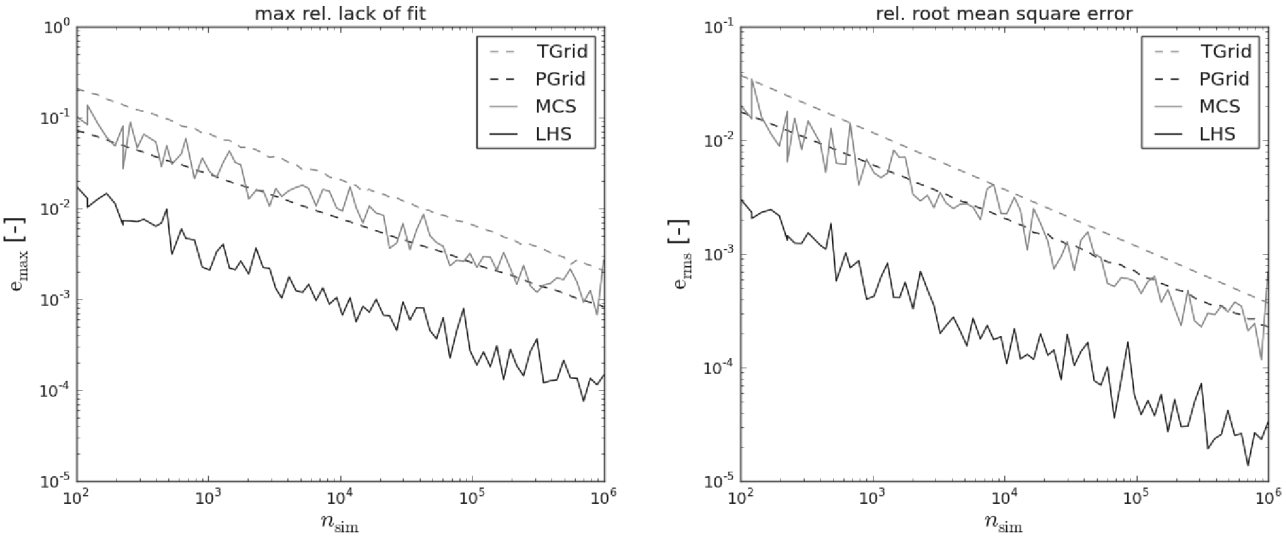


Figure 4: Convergence to an exact solution with an increasing number of sampling points in terms of local and global error measures given in Eqns. (9) and (10), respectively.

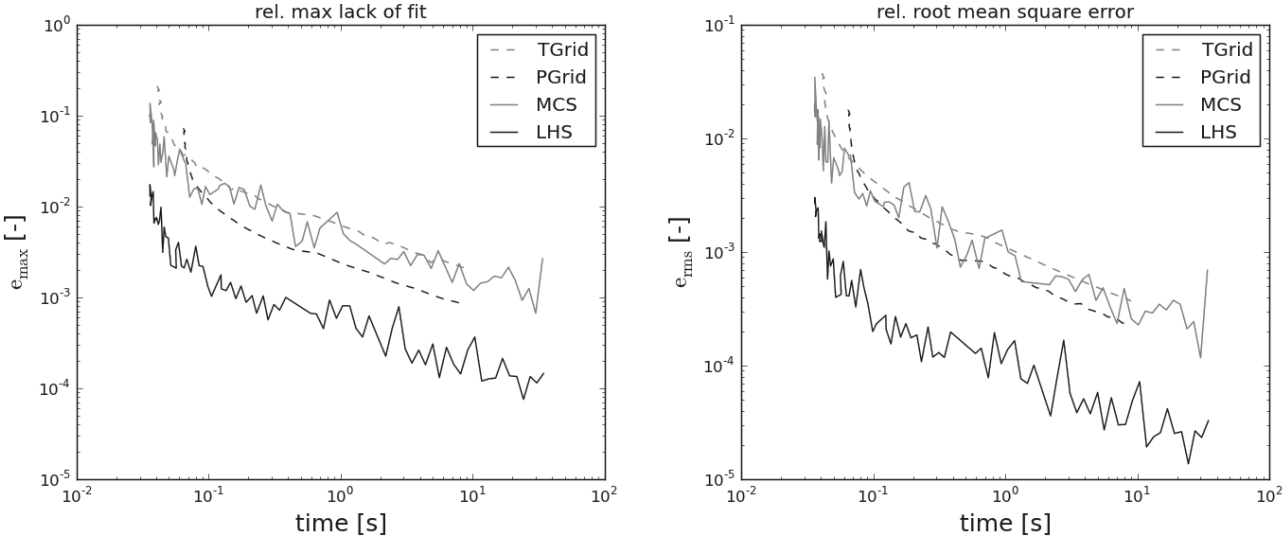


Figure 5: Convergence to an exact solution with an increasing computational time in terms of local and global error measures given in Eqns. (9) and (10), respectively.

where $i = 1 \ldots n_{\varepsilon}$. Fig. 4 shows the convergence behavior in double logarithmic scale for both types of errors for an increasing number of sampling points. Both diagrams document the fact that LHS covers the random domain in a significantly more efficient way than all the other implemented methods. The convergence behavior of the other three sampling methods is comparable.

A more relevant comparison of efficiency is given in Fig. 5 which shows the measured CPU time for the studied sampling methods instead of n_{sim}. Also in this view, the LHS method is superior to the other methods. PGrid sampling is revealed to be slightly more efficient than Monte-Carlo and TGrid. The higher efficiency is due to the fact that during the response function evaluation, some interim results can be reused in the broadcasted dimension and, consequently, a significant number of operations can be saved. Such a caching of interim values within a random subspace is not possible in the Monte Carlo type of sampling due to the unstructured coverage of the m-dimensional space. The positive effect of vector-

ized evaluation increases as the number of random parameters grows. However, it also depends on the type of the response function and on the current choice of the random parameters. It must be also considered that the vectorized evaluation of the response function $q(\varepsilon)$ in the m-dimensional space is connected with the exponential growth of memory consumption.

5 Execution efficiency of vectorized and compiled code

Previous studies shown in Fig. 5 reveal that the CPU time required to achieve sufficiently accurate results ($\mathrm{e}_{\mathrm{rms}} \leq 10^{-3}$) for a function with two random variables using the numpy code is less than 1 sec. For more complex functions with more random variables further speedup might be desirable. There are several ways to improve the execution efficiency of Python code. An overview of speedup possibilities for numerical Python code has been provided in [13].

In order to accelerate the present application two options have been used: cython [14] and weave [15]. They have been incorporated into the spirrid package as subclasses of the CodeGen class (see Fig. 2). Studies of speedup achievable using the weave and cython packages have been implemented as a script interacting with a single SPIRRID instance. Execution times have been measured for the two-parametric function in Eq. (1) of our running example with two types of sampling: LHS and PGrid. The size of the sample for both sampling schemes was chosen in order that a comparable level of accuracy was reached based on the studies of sampling efficiency shown earlier in Fig. 5. The accuracy required for the studies was $\mathrm{e}_{\mathrm{rms}} \leq 5 \cdot 10^{-5}$, which corresponds to $n_{\mathrm{sim}} = 440^2$ for LHS and $n_{\mathrm{sim}} = 5000^2$ for PGrid. CPU time was measured for three phases of the computation:

- the sampling time needed for preparing the arrays $\boldsymbol{\theta}$ and $\Delta \boldsymbol{G}$ within the sampling object,
- the time needed to prepare and compile the method mu_q_method within the codegen object, and
- the time needed to execute the integration procedure on the prepared data arrays.

Each version of the code was executed twice in order to show the difference between CPU time with and without the setup and compilation of the generated C code. CPU times obtained for LHS shown in Fig. 6 (left) reveal that a significant amount of time was spent on the permutation of $\boldsymbol{\theta}$ arrays. For all three types of code a standard permutation algorithm available in the numpy package was used so that the sampling time remained constant (≈ 0.19 s) in all runs.

The shortest execution time was achieved in the second run of the compiled weave code of the code was only insignificantly slower. This is not surprising since both compiled versions lead to an optimized C code with a similar structure. As already stated in [13] the efficiency of weave and cython can be regarded as equivalent. The overall CPU time of the scripting numpy version was about 2.5 times longer than for both compiled versions. Regarding the time required by the pure integration procedure, the compiled code is 12 times faster than the scripting code. This relation is in agreement with the studies published in [13].

Even though the grid-based sampling schemes are significantly slower than LHS it is interesting to examine the effect of compilation on the speedup for the PGrid sampling shown in the right diagram of Fig. 6. Regarding the CPU time required by the pure integration (numpy: 18.44 s, weave: 3.57 s) we can see that the speedup factor (≈ 5) is much smaller than in the case of LHS (≈ 12).

In the simple example used for this study, the setup and compilation time of the first run was longer than the time spent on computation. For larger problems with a more complex response function, more random variables and more sampling points, the proportion of time spent on setup and compilation would certainly diminish. Studies of more complex functions with different combinations of random variables have been included in the spirrid package [2].

6 Conclusion

In the present paper we have reported on the scientific computing tool for estimation of statistical characteristics of multi-variate random functions developed in high-level programming language Python. We have described mathematical formulation of the problem, designing of an algorithmic object reflecting the state dependencies between the editable components and speeding up of the code to achieve the efficiency of low-level compiled code. The resulting algorithmic object for statistical integration can be interactively edited by modifying the function, declaring its parameters as control or random variables, choosing and configuring probabilistic distributions of the random variables, selecting from four types of sampling schemes and configuring the execution code for the integration.

The code of spirrid package based on the Enthought Traits library has been made available for downloading through the Github repository [2].

ISBN: 978-1-61804-117-3

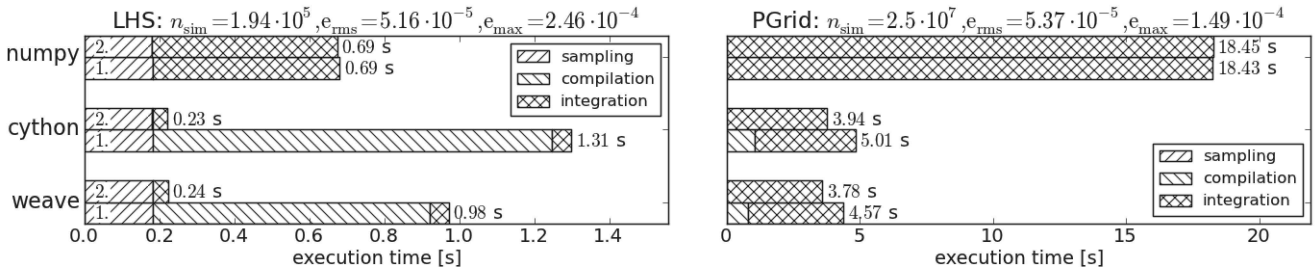


Figure 6: Comparison of execution times for LHS and PGrid sampling needed in the first and second run of numpy, cython and weave code ($n_\varepsilon = 80$).

Acknowledgements: The work of the first and second author was supported by the Czech Science Foundation under project no. GCP105/10/J028, by project FAST-S-12-1804 and by the project CZ.1.07/2.3.00/30.0005 of Brno University of Technology. The work of the third and fourth author was supported by the Deutsche Forschungsgemeinschaft (DFG project number CH/276/2-1 and SFB532/T05). This support is gratefully acknowledged.

References:

[1] R. Chudoba, V. Sadílek, R. Rypl, and M. Vořechovský. Using Python for scientific computing: an efficient and flexible evaluation of the statistical parameters of functions with multivariate random inputs. *Journal of Computer Physics Communications*, In review.

[2] SPIRRID: Tool for estimation of statistical characteristics of multi-variate random functions, http://github.com/simvisage/spirrid, 2011.

[3] H. P. Langtangen, X. Cai, On the Efficiency of Python for High-Performance Computing: A Case Study Involving Stencil Updates for Partial Differential Equations, in: H. G. Bock, E. Kostina, H. X. Phu, R. Rannacher (Eds.), Modeling, Simulation and Optimization of Complex Processes, Springer Berlin Heidelberg, 2008, pp. 337–357.

[4] S. v. d. Walt, S. C. Colbert, G. Varoquaux, The NumPy array: A structure for efficient numerical computation, Computing in Science & Engineering 13 (2011) 22–30.

[5] J. Nilsen, MontePython: implementing quantum Monte Carlo using Python, Computer Physics Communications 177 (2007) 799–814.

[6] Z. Kala, Sensitivity analysis of steel plane frames with initial imperfections, Engineering Structures 33 (2011) 2342–2349.

[7] Z. Kala, Geometrically non-linear finite element reliability analysis of steel plane frames with initial imperfections, Journal of Civil Engineering and Management 18 (2012) 81–90.

[8] S. L. Phoenix, H. M. Taylor, The asymptotic strength distribution of a general fiber bundle, Advances in Applied Probability 5 (1973) 200–216.

[9] R. Chudoba, M. Vořechovský, M. Konrad, Stochastic modeling of multi-filament yarns I: Random properties within the cross section and size effect, International Journal of Solids and Structures 43 (2006) 413–434.

[10] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley Professional, 1st edition, 1998.

[11] D. C. Morrill, J. M. Swisher, Traits 4 user manual, http://docs.enthought.com/traits/traits_user_manual/index.html, 2011.

[12] L. Pierce, J. Swisher, Traits UI 4 user manual, http://docs.enthought.com/traitsui/traitsui_user_manual/index.html, 2011.

[13] I. M. Wilbers, H. P. Langtangen, A. Odegard, Using Cython to speed up numerical Python programs, Proceedings of MekIT 9 (2009) 495–512.

[14] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, K. Smith, Cython: The best of both worlds, Computing in Science & Engineering 13 (2011) 31–39. WOS:000288053300004.

[15] Weave: tools for inlining C/C++ within Python code, http://www.scipy.org/Weave, 2011.