Structural and Multidisciplinary Optimization (2019) 60:405–409
https://doi.org/10.1007/s00158-019-02298-4

EDITORIAL

Replication of results

Raphael T. Haftka[1] · Ming Zhou[2] · Nestor V. Queipo[3]

Received: 28 January 2019 / Revised: 18 April 2019 / Accepted: 30 April 2019 / Published online: 3 June 2019

There is a growing movement to make scientific and engineering research more useful by making it easier for potential users to access papers and to replicate their results. Structural and Multidisciplinary Optimization (SMO) has joined this movement by allowing authors to pay for open access of their papers and by recently requiring that papers have a section called Replication of Results (RR) intended to facilitate the reproduction by readers of the presented results. The phrase RR makes reference to the idea of providing a detailed enough description of the proposed model/method so that the results can be replicated to a degree consistent with the claims and conclusions drawn in the paper (Plesser 2018). This editorial has two objectives: (i) to outline the benefits to the community and the authors of the RR section and (ii) to present options available to authors to facilitate replication of results.

There is no consensus on terminology. As Plesser (2018) notes, the alternative term is reproduction of results. Indeed, when authors appear not to understand the purpose of this section, the first author of this editorial often writes to them that the section is intended to help readers reproduce the results presented in the paper.

Almost all the papers in SMO are about computational algorithms, and so this editorial is about computational replication of results. There is substantial literature on tools and techniques for this purpose, see a good review by Piccolo and Frampton (2016). Some of the literature is focused on what is needed for exact replication, and this is a major challenge because readers may be implementing algorithms in different computer languages on different computer architectures, and also use different numerical libraries. Here, we also stress the importance of authors to probe the variability of the algorithm, so that readers will know how much their results may be expected to vary from those published in the paper and yet remain consistent.

1 Benefits

Benefits to the community The RR section should help good stewardship of the research record and reduction of research waste, with potentially better use of funding across the community. It will allow the quicker implementation of new methodology because developers could readily benchmark against original reference code by comparing results of examples presented in related papers. Similarly, the section could also accelerate the development of better methods building on existing work because it would help verification of and comparison with existing methods. It is our observation that many papers tend to forgo details on the implementation of formulations presented. This makes it very difficult for a reader to attempt implementing methods presented in such papers, let alone reproduce results. The requirement of the RR section also implies that authors should reveal all details about transferring their theory into computer code.

Benefits to the authors and journal The success of researchers and journals is greatly influenced by the impact of their papers as measured by citations, downloads, and reads. From our experience, papers that help the reader reproduce their content are more likely to be read, cited and encourage implementation, and use of the proposed methods. Furthermore, the authors themselves often need to continue research that they documented in papers years before, and they benefit from their own ability to reproduce the results of old papers. This is particularly important as many of SMO's papers are published by authors from research groups that persist over decades. Such research teams at universities are typically dynamic, which renders it necessary for new members to carry on a research code for continuing studies.

Responsible Editor: Hyunsun Alicia Kim

✉ Raphael T. Haftka
haftka@ufl.edu

[1] University of Florida, Gainesville, FL, USA

[2] Altair, Irvine, CA, USA

[3] University of Zulia, Maracaibo, Venezuela

Another benefit of documenting all ingredients for results is that it reinforces the importance of developing robust methods. The following process may look familiar to many readers. Starting with the first example, it typically requires tens or even hundreds of experiments until satisfactory results are obtained. This process typically involves (1) fixing bugs in the code and (2) testing parameters within the formulation (e.g., penalty, minimum density threshold for topology optimization) and parameters for the iterative process (e.g., variable move limits, convergence tolerance, or a number of maximum iterations). The best result is selected at the end of the process. The researcher then moves on to the second example. Again, many results are produced until he/she is satisfied with some results. This goes on until results are finalized for all selected examples. There is nothing wrong with the process if it was aimed at producing a bug-free code and selecting a robust parameter setting. In other words, results selected for publication should be all produced with the final version of the code, with as little changes of default parameters as possible. Note that different parameters typically result in different results. The selection of default parameters like convergence criteria aims at obtaining reasonable results for typical problems. If the method requires tuning of parameters for each example, it would not be a viable method. The author would not be able to miss such vulnerability of a method when detailed settings for results need to be documented. Similarly, selective reporting can be a serious problem for research publications. However, such selective reporting is mitigated when it is easy for readers to try their own examples.

2 Structuring research so that it is easily reproduced

The first step in helping oneself and readers to reproduce results should be taken in the way the research is implemented. This has received much attention in the biological community, see Markowetz's Five Selfish Reasons to Work Reproducibly (Markowetz 2015). Consider the following issues:

Application examples Sometimes research is motivated by complex real-life applications, which would be difficult to replicate even if the authors were free to share all the needed data and codes. In that case, it is desirable to look for simpler examples with similar characteristics. For example, Barthelemy and Haftka (1990) became aware of the accuracy problems of the semi-analytical method for shape sensitivity of static response when calculating derivatives of car finite element models. They looked for a simple problem that would illustrate the difficulty and found that it could be reproduced with a cantilever beam. Simple examples are also useful for the authors to validate their own analysis code, by replicating their results on widely available commercial codes that may have a free version with model size limitation.

An alternative approach is to create examples based on a surrogate fit to simulations from proprietary codes. For example, Guo et al. (2018) became aware of difficulties with choosing between single- and multi-fidelity surrogates for a turbine problem. They found that they could replicate the difficulties by replacing the turbine efficiency response by a cubic polynomial in terms of the two geometry design variables, and they provided the polynomial in the publication.

Randomness Many research endeavors require manipulations of random numbers. Papers on reliability calculations and design under uncertainty often require Monte Carlo simulations. Many global search algorithms such as simulated annealing or genetic algorithms have multiple calls to random number generators. Surrogate fitting is often performed with Latin hypercube sampling that leads to somewhat random designs of experiments.

Iterative processes with convergence criteria are often a source of randomness because on one computer system, the criterion may be barely satisfied and on another barely missed. Optimization algorithms typically have such convergence criteria. For example, Kim et al. (2001) report on variability created by optimization algorithms and propose a technique to detect poorly converged runs. Distributed computing is also often a source of randomness.

Fixing the random numbers can be done by selecting a particular seed for the random number generator, as with the rng function in Matlab and the random.seed function in Python. This has the extra benefit for debugging the research code because results become repeatable.

There are technological solutions to overcome randomness generated by changes in computers or their operating systems. These require more than simply documenting the data and software used. Specifications of the operating system and computer configuration, and how to install the necessary software dependencies should also be provided. Virtual machines, for example, can help address this issue by encapsulating an entire operating system and all scripts, code, and data necessary to execute a computational analysis. A virtual machine (constructed with tools such as VirtualBox or VMWare) can be executed on practically any desktop, laptop, or server, irrespective of the host operating system. A lighter alternative to virtual machines is the so-called software containers (e.g., open source Docker.com utility). While these containers are specific to a given type of operating system and may be more vulnerable to security breaches, they have less computational overhead than VM and can be set up more quickly.

1 Example of fitting support vector regression (SVR) to the Branin-Hoo function

In [1]:
```
%matplotlib inline
# conda install -c conda-forge pydoe (should be executed from the Anaconda command prompt)
from matplotlib import cm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from pyDOE import lhs
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

# Random seed initialization
np.random.seed(1)
```

1.1 Test function

In [2]:
```
# Defining the Branin Hoo function
BH = lambda X: (X[:,1] - 5.1/4*(X[:,0]/np.pi)**2 + 5*X[:,0]/np.pi - 6)**2 + \
  10*(1-np.pi/8)*np.cos(X[:,0])+10
```

1.2 Sample generation

In [3]:
```
# Generating 20 lhs samples for training in x1 [-5,10], x2 [0,15]
X = (15*lhs(2,samples=20)) - [5, 0]
y = BH(X)
# Generating 200 lhs samples for testing
Xtest = (15*lhs(2,samples=200)) - [5, 0]
ytest = BH(Xtest)
CST = {'case': "Branin-Hoo", 'X': X, 'y': y, 'Xtest': Xtest, 'ytest': ytest }
```

1.3 Grid search

In [4]:
```
# Parameters for GridSearchCV
PARS = {'C': [55.0, 110.0, 220.0], 'epsilon': [0.1, 0.2, 0.3], 'gamma': [0.01, 0.1, 0.2]}

MOD = GridSearchCV(estimator=SVR(kernel='rbf'), param_grid=PARS, cv=5, refit=True)
MOD.fit(CST['X'], CST['y'])
```

Out[4]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'C': [55.0, 110.0, 220.0], 'epsilon': [0.1, 0.2, 0.3], 'gamma': [0.01, 0.1, 0.2]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

Fig. 1 Jupyter notebook example

Exploring variability and robustness Papers proposing new or improved algorithms and papers reporting on experience with existing algorithms are most useful when their conclusions are robust with respect to whatever source of randomness. So authors should make repeated runs in order to determine the scatter in the results due to the randomness in the process. If the paper is developing an improved method and the answer is random, comparison with competing techniques would depend on the mean or median of the results as well as their variability. In such scenarios, whenever possible, statistical significance tests should be conducted on the differences observed indicating p values, e.g., Mood's median statistical test.

The randomness associated with operating systems may often be imitated by small perturbation in loading, finite element meshes, or material properties. This is due to the fact that all of these will operate on randomness associated with convergence criteria of iterative processes.

3 Options for providing useful information

Source code Providing the reader with the source code used in the paper is the best option. If it is short, it can be included in an appendix, otherwise as supplementary material or a link for downloading the code. Another alternative is to store them in a public repository using a version control system and to share it via web-based services like GitHub.com or Bitbucket.org. Then, readers can see the full version history, reuse the code, and contribute revisions to the code (if applicable). When

1.4 Performance indicators

In [5]:
```
results = dict()
results['R2']    = MOD.score(CST['X'], CST['y'])
results['MSEtr'] = mean_squared_error(CST['y'], MOD.predict(CST['X']))
results['MSEte'] = mean_squared_error(CST['ytest'], MOD.predict(CST['Xtest']))
print("%25s\t%10s\t%10s\t%10s" % ("Surrogate", "R2", "MSE(train)", "MSE(test)"))
print("%25s\t%10.3f\t%10.3f\t%10.3f" % ('SVR', results['R2'], results['MSEtr'], results['MSEte']))
print("%25s\t%s" % ("Best parameters", MOD.best_params_))
```

```
                Surrogate              R2      MSE(train)       MSE(test)
                      SVR           0.954         148.688         204.199
          Best parameters       {'C': 220.0, 'epsilon': 0.1, 'gamma': 0.01}
```

1.5 Plots

In [6]:
```
# Grid for visualization
X1,X2 = np.meshgrid(np.arange(-5.0, 10.0, 0.5), np.arange(0.0, 15.0, 0.5))
XX1 = X1.flatten(); XX2 = X2.flatten()

# SVR prediction
YY  = MOD.predict(np.c_[XX1, XX2])
Y = YY.reshape(X1.shape)

# Plotting training points and SVR prediction surface on grid
fig = plt.figure(figsize=plt.figaspect(0.35))
ax = fig.add_subplot(1, 2, 1, projection='3d')
ax.plot_surface(X1, X2, Y, cmap=cm.coolwarm, alpha=0.75, rstride=1, cstride=1)
ax.scatter(CST['X'][:,0], CST['X'][:,1], CST['y'], c='r', s=50)
plt.xlabel('X1'); plt.ylabel('X2'); plt.title('SVR prediction surface')
ax.set_zlabel('Y'); ax.axis('equal'); ax.axis('tight'); ax.set_zlim(0,300)
ax.view_init(30,-130)

# Real function evaluation on grid
YY_real = BH(np.c_[XX1, XX2])
Y_real = YY_real.reshape(X1.shape)

# Plotting real function surface
ax2 = fig.add_subplot(1, 2, 2, projection='3d')
ax2.plot_surface(X1, X2, Y_real, cmap=cm.coolwarm, alpha=0.75, rstride=1, cstride=1)
plt.xlabel('X1'); plt.ylabel('X2'); plt.title('Real function surface')
ax2.set_zlabel('Y'); ax2.axis('equal'); ax2.axis('tight'); ax2.set_zlim(0,300)
ax2.view_init(30,-130)

plt.show()
```

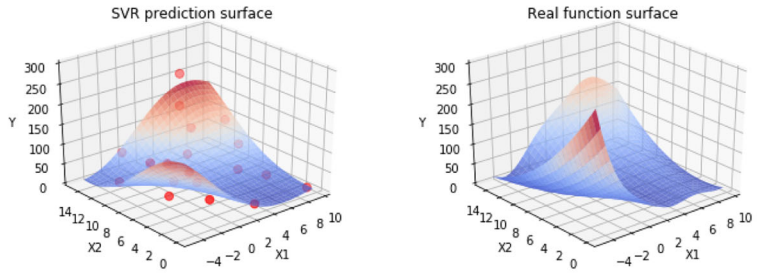

Fig. 1 (continued)

submitting a manuscript, the authors may "tag" a specific version of the repository that was used for the final analysis described in the manuscript and assign to it as permanent digital object identifiers (DOI), through, for example, Zenodo.org or figshare.com.

It helps the reader if the code is heavily commented, even if this increases its length. Literate programming (e.g., Piccolo and Frampton 2016) pushes documentation to a high level. It produces notebooks that allow to create and share documents that contain live code, equations, visualizations, and narrative text; Jupyter and knitr are popular examples of this new computing paradigm. Cloud services where these notebooks can be run without any setup are increasingly available, e.g., Jupyter notebooks in Colaboratory by Google. An example of a code that fits the Branin-Hoo functions by support vector regression in Fig. 1. As can be seen, the example not only has the commented code, but it also shows the numerical and graphical results.

Also, it helps to put all the tuning or convergence parameters at the beginning of the code or to color the statements that include them. Consider, for example, what may be the most heavily used code ever provided in SMO, Sigmund's 99 line Matlab

topology optimization code (Sigmund 2001). Lines 39 and 40 of the code read

```
l1 = 0; l2 = 100,000; move = 0.2;
while (l2-l1 > 1e-4).
```

The three numbers selected by the author (100,000, 0.2, and 1e-4) have been changed for some problems in the first author's classes. With a 99 line code, it is very easy to spot such choices of convergence or tuning parameters, but with a much longer code, this may be a problem.

Executable code When the source code cannot be provided, as for example, due to institutional constraints, an executable version of the code may still be useful. For replication of results, users can benefit from an executable code if there is a flag that can be tripped to provide intermediate results that would help a reader debug their own code for the algorithm proposed in the paper. Such executable code would also be useful for a reader that may want to test the method for the solution of other problems.

Pseudocode Pseudocodes for the algorithms can be very useful, especially if they also mention all tuning and convergence parameters.

Input files and scripts for commercial codes When the results are generated using commercial codes, the input files and scripts used with these codes should be provided if possible.

Digital data for plots Often results are presented in plots, and readers who want to compare to their results need to recreate these plots. It is useful then to provide, as supplementary material, the numerical data used to create the plots.

Detailed intermediate results For iterative algorithms, such as optimization, providing details on design data at every iteration is an example of details that may help readers debug their replication of the results. Such information often does not belong in the paper itself but is natural for supplementary material.

4 Replication of results

The only results presented in this editorial are in Fig. 1. The source code is part of that figure.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

Barthelemy B, Haftka RT (1990) Accuracy analysis of the semi-analytical method for shape sensitivity calculation. Mech Struct Mach 18(3): 407–432

Guo Z, Song L, Park C, Li J, Haftka RT (2018) Analysis of dataset selection for multi-fidelity surrogates for a turbine problem. Struct Multidiscip Optim 57(6):2127–2142

Kim H, Papila M, Mason WH, Haftka RT, Watson LT, Grossman B (2001) Detection and repair of poorly converged optimization runs. AIAA J 39(12):2242–2249

Markowetz F (2015) Five selfish reasons to work reproducibly. Genome Biol 16:274

Piccolo SR, Frampton MB (2016) Tools and techniques for computational reproducibility. GigaScience 5(1):30

Plesser HE (2018) Reproducibility vs. replicability: a brief history of a confused terminology. Front Neuroinform 11:76

Sigmund O (2001) A 99 line topology optimization code written in Matlab. Struct Multidisc Optim 21:120