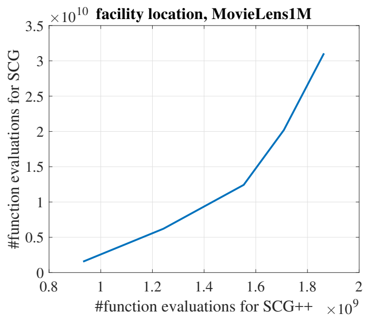
We thank the reviewers for their careful consideration and constructive feedback. Below, please find our responses.

General response to all reviewers regarding empirical study of SCG++. This paper is primarily of theoretical nature and aims to provide a complete answer to an open question regarding the sample complexity of DR-Submodular maximization with stochastic oracles. However, the main suggestion of all the reviewers for improving the paper has been to provide an empirical study on SCG++ and compare its performance/complexity with the existing methods such as stochastic continuous greedy (SCG). We will hence provide a separate section in the revised manuscript that provides a thorough empirical study on the performance of SCG++ by comparing it with the state-of-the-art methods using practical applications. Moreover, in order to highlight the superiority of our method (SCG++) compared to the existing methods, we have implemented both SCG++ and SCG on a real-world instance of (stochastic) discrete submodular optimization (see equation (20) in the paper). The setting involves selecting a subset of movies to recommend to users based on previously obtained data (ratings) from the users (we have used the MovieLens data set with 1M entries). In the plot provided above, we let the x-axis (resp. y-axis) represents the amount of function evaluations that SCG++ (resp. SCG) needs to achieve the *same* solution quality. In other words, every point in the plot corresponds to a specific solution quality and represents the total number of function evaluations needed by SCG++ (the x-coordinate of the point) as well as SCG (the y-coordinate) to achieve that solution quality. As we observe from the plot, SCG requires a higher number of function evaluations with respect to SCG++ and the difference diverges as the solution quality increases.



Response to Reviewer #1. Q1: My primary concern with the paper is how much these results add value in a practical setting. Since no experimental evidence are provided... **A1:** Please see our general response above. We would also like to emphasize that, as proven in the paper, SCG++ beats the existing methods in every aspect. Hence, we expect to observe a similar superiority in our numerical simulations (we have provided one example in the plot above).

Q2: If 'F' is available in closed form and the its gradients can be computed exactly.... **A2:** If the gradients can be computed exactly then no variance reduction is needed (i.e. if there is no stochaticity/randomness, then there is no variance to be reduced). In this case, both SCG and SCG++ reach a $(1 - 1/e - \epsilon)$-opt solution with $O(1/\epsilon)$ oracle calls.

Response to Reviewer #2. Q3: Improve the readability. **A3:** We will provide further explanation in the main part of the revised version about the ideas behind our methods and will include proof sketches. Thanks for your suggestions.

Q4: Extension to the discrete setting: I do not understand how one would compute the multilinear extension efficiently. **A4:** We do not need to compute the exact value of the multilinear extension; We only need an unbiased estimate of the value of the multilinear extension which can easily be obtained using a single function evaluation. More precisely, the multilear extension is defined as $F(\mathbf{x}) = \mathbb{E}_{S\sim\mathbf{x}}[f(S)]$ and an unbiased estimate of $F(\mathbf{x})$ can be obtained by generating a random subset $S \sim \mathbf{x}$ and computing the corresponding function value $f(S)$ as the estimate. In a similar manner, we can obtain an unbiased estimate of the partial derivatives as well as the entries of the Hessian of the multilinear extension using a few function evaluations (as explained in detail in lines L272-280 of the paper). These unbiased estimates are the plugged into our proposed stochastic algorithms to provide the desired solutions. Theorem 2 provides the exact sample complexity (i.e. the number of function evaluations) of our proposed algorithms. Also, Sections 8.4-8.6 in the appendix provide a detailed explanation on how to implement our methods in the discrete setting).

Q5: Lack of concrete examples: What are concrete examples of continuous submodular functions and how do the results in this paper impact them? **A5:** We will address this comment thoroughly in our revised version. In brief, continuous submodular functions generalize the notion of diminishing returns to continuous domains. They have recently gained considerable attention and have been used in applications such as inference in determinental point processes, resource allocation, experimental design, etc. Moreover, the multilinear extension of any discrete submodular function is a continuous submodular function. The main impact of our algorithms is in the so-called stochastic setting: Oftentimes in practice, the function value (or its gradient) can not be computed exactly and we need to consider (stochastic) estimates. For example, in large scale-settings, computing the function (or its gradient) could amount to a full pass over the data which is an expensive task and hence we resort to stochastic min-batches to speed up computation instead of a full pass over data. Indeed, stochastic optimization techniques are among the key contributors to the success of large-scale machine learning. Our paper provides algorithms for stochastic submodular optimization with optimal sample complexity (i.e. the minimum number of queries from the stochastic oracle).

Q6: Lack of empirical results demonstrating the improved convergence. **A6:** Please check our general response above.

Response to Reviewer #3. We would like to thank the reviewer for the comments and suggestions. About the empirical experiments, please see the general response given above.

Q7: The proposed bound needs the smoothness of the Hessian , and I think without such an assumption, it would require $O(d^2/\epsilon^2)$ oracle calls. **A7:** The reviewer is right that if we do not assume Hessian smoothness, then for running SCG++ we need overall $1/\epsilon^2$ stochastic Hessian approximations which means that the overall computational complexity of SCG++ will be d^2/ϵ^2. We will highlight this point in the revised version.