. . . continued from Exercises 2

Stochastic T-IFISS extends the core version of T-IFISS to cover stochastic Galerkin approximations of diffusion problems with random coefficients, the associated a posteriori error estimation and adaptive algorithms, including goal-oriented adaptivity. It can be downloaded from
`http://web.mat.bham.ac.uk/A.Bespalov/software/`

10. The aim of this exercise is to assess the effectiveness of the default adaptive refinement strategy that is built into stochastic T-IFISS by looking at a problem with a regular solution. The test problem can be set up in stochastic T-IFISS by running the driver `stoch_adapt_testproblem` and selecting reference problem 2. Taking all the default choices, you should discover that the adaptive algorithm converges in 47 steps with 4 parametric enrichments. The total number of degrees of freedom should be 109136 with 7133 vertices on the final mesh with 16 indices activated and 5 active parameters. Save the plots of the refinement path (Figure 120) and of the final mesh (Figure 2). (Hint: use the command `savefig`.)

    Next, repeat the experiment by running `stoch_adapt_diff_main` with the hierarchical strategy EES^2 together with edge marking. You should observe that far fewer adaptive steps are needed to reach the default tolerance and the number of vertices on the final mesh is reduced by a factor of close to 2.

    Another point worth noting is that, in both cases, the order of convergence is close to $N^{-1/3}$ where N is the number of degrees of freedom, which is suboptimal.

11. The aim of this final exercise is to assess the role of the marking threshold parameters on the adaptive algorithm efficiency. To this end, you might consider repeatedly solving the singular problem given by running `stoch_adapt_testproblem` and selecting reference problem 5. Choose the default error tolerance and run the code with the default strategy EES^1 with element marking and select a fast decaying random coefficient series representation.

    Generate a two-dimensional table of results showing iteration counts and associated solution times for 3 different values of each marking parameter (9 runs of the code in total). You should discover that the default choices $\theta_x = 0.2$ and $\theta_y = 0.9$ are close to optimal in terms of the cpu time!