Exercise 5.2.3: Optimization, Search & Exploration 3

In this exercise set, you will see:

- The use of a stochastic search algorithm: PGSL.

Introduction to PGSL

Global optimization techniques, such as stochastic search, are powerful techniques for complex engineering tasks. When there are many local minima, they find solutions that have a greater chance of being the global optimum than other methods such as gradient search. The optimization technique used in these exercises, PGSL (Probabilistic Global Search Lausanne), is based on the assumption that sets of near-optimal solutions are more likely to be found near sets of good solutions. Search is driven by a probability density function that is iteratively modified so that more exhaustive searches are made in regions of good solutions (Raphael and Smith, 2003). Solutions depend on problem parameters that are defined through lower and upper bounds of possible values. Finally, solutions are identified based on the objective function, which may contain penalties if constraints are included.

The PGSL algorithm is available for free downloading at:

Exercise 1: Optimization of a floor surface

An engineer needs to design a floor surface for a public area. He selects a system made of simple beams (see figure below) on which there is a uniform load, q. The beams are regularly spaced at a distance “e” (see Section A-A). Section B-B shows the end conditions of the beams and Section C-C is their cross-section.
The beams are to be sized using the following relations and constraints:

**Relations**

<table>
<thead>
<tr>
<th>Relation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \delta = \frac{5qL^4}{384EI} ]</td>
<td>Deflection at half span of a simple beam under the effect of a load, q.</td>
</tr>
<tr>
<td>[ \sigma_{\text{max}} = \frac{M}{W} ]</td>
<td>Maximum constraint (avec ( W = \frac{I}{(d/2)} ))</td>
</tr>
<tr>
<td>[ M = \frac{qet^2}{8} ]</td>
<td>Maximum moment at half span of a simple beam under the effect of uniform loading, q · e.</td>
</tr>
</tbody>
</table>

**Constraint**

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \delta_{\text{selfweight}} + \delta_{\text{uniformload}} \leq \delta_1 = \frac{L}{250} ]</td>
<td>Servicability Criterion</td>
</tr>
<tr>
<td>[ \delta_{\text{uniformload}} \leq \delta_2 = \frac{L}{350} ]</td>
<td>Rigidity Criterion</td>
</tr>
<tr>
<td>[ \sigma_{\text{max}} \leq f_y ]</td>
<td>Allowable Strength Criterion</td>
</tr>
<tr>
<td>[ d \leq \frac{L}{10} ]</td>
<td>Criterion to validate simple beam bending theory</td>
</tr>
<tr>
<td>[ 0.5 \leq e \leq 3.0 ]</td>
<td>The spacing of the beams needs to be between 0.5 and 3m for practical questions (min value) and floor strength (max value)</td>
</tr>
<tr>
<td>Length = 50 m</td>
<td>The length of the floor is fixed by the architect at 50m.</td>
</tr>
<tr>
<td>q = 4 kN/m²</td>
<td>The uniform load is defined by the code.</td>
</tr>
</tbody>
</table>

The optimized supporting structure is composed of square wooden beams.
Characteristics of the square wooden sections:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Width [m]</td>
<td>$b \leq 0.24$</td>
</tr>
<tr>
<td>d</td>
<td>Height [m]</td>
<td>$d \leq 0.3$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Volumetric Weight [kN/m$^3$]</td>
<td>$\gamma \leq 5.0$</td>
</tr>
<tr>
<td>$f_y$</td>
<td>Yielding limit [kN/m$^2$]</td>
<td>$f_y = 1e4$</td>
</tr>
<tr>
<td>$E_b$</td>
<td>Youngs Modulus [kN/m$^2$]</td>
<td>$E_b = 1e7$</td>
</tr>
<tr>
<td>$CU_b$</td>
<td>Unit Cost [$/m^3$]</td>
<td>100.00</td>
</tr>
</tbody>
</table>

a. An engineer proposes beams with dimensions of 18 by 30cm, spaced 75cm apart over a span (L) of 6m. Manually verify if these dimensions are acceptable. Calculate the total price.

b. Create a program that uses the PGSL algorithm with 5000 iterations that allows for the dimensions of the floor structure to be calculated in order to minimize the total price (in $). Assume that the assembly characteristics do not have an effect on the price (incorrect in practice). Assume all beams have similar dimensions. Appendix A shows an example java script which calls the PGSL algorithm. The PGSL algorithm is available in C-compatible programs and Matlab format. (The solution to this question will show an example written in Java.)

c. Test your program with the numerical values from the table below and determine the optimal numerical values of the wooden sections (justify):

<table>
<thead>
<tr>
<th>Case</th>
<th>L [m]</th>
<th>q [kN/m$^2$]</th>
<th>Optimal Solution(s)</th>
<th>Price</th>
<th>Active Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>b [m]</td>
<td>d [m]</td>
<td>e [m]</td>
</tr>
<tr>
<td>1</td>
<td>6.0</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8.0</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10.0</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

d. What is the difference in % of the total price between the dimensions in a and the dimensions optimized in c, for the first situation?

**Exercise 2: Probabilistic Global Search Lausanne (PGSL)**

Answer the following questions (with justification):

a. If you do not know the behavior (form) of the function to be minimized, with which method would you start?
b. How can you be sure that this method is reliable?

c. Give an example of a situation where you would not use the PGSL method.

d. Is there a means to obtain exactly the same result twice with PGSL?

Acknowledgements


I.F.C.Smith, 2011
Appendix A

The program below illustrates how to implement an optimization problem using the PGSL algorithm. It is sufficient to give the object function (cost function) and the bounds of the varying parameters that are to be optimized. In the example below, the objective function is a parabolic function \( a \times x_1 + b \times x_1 + c \) and the bounds for the varying parameter \( x \) is between -100 and 100.

File used: SPGSLExample.java

SPGSLExample.java

```java
import SPGSL.*;   // import the package containing the algorithm
import java.text.*;

public class SPGSLExample {
    static DecimalFormat df = new DecimalFormat();

    /* Specification of the number of variables */
    static int numVariables=1;

    //-------------------------------------------------------

    /* creating a new class derived from the class problem */
    public static class SampleProblem extends Problem {
        int numTrials;
        double a = 2;
        double b = -6;
        double c = 3;

        public SampleProblem(int numVars, long numEvaluations) {
            super(numVars, numEvaluations);  /* call the constructor of the parent class */

            double [ ] min= { -100};
            double [ ] max= { 100};
            int i;

            for (i=0; i<numVars; i++) {
                axes[i] = new SPGSL.Axis(min[i], max[i], 1);
            }

            this.threshold = -1000;  /* threshold at which solver stops */
        }

        //-------------------------------------------------------

        /* The objective function has to be defined */
        public double costFunction(double [ ]paramValues) {
            double value=0;

            // Find the value of the parameter that will be obtained by the algorithm.
            double x1 = paramValues[0];

            /* Calculate the value of the objective function */
            value = a * x1 * x1 + b * x1 + c;
        }
    }
}
```
/* State the number of trials */
numTrials++;

/* Give the results of the evaluation */
System.out.println("\t" + numTrials + "\t" + df.format(x1)+"\t" +
    df.format(value));

return value;
}

//------------------------------------------------------
}

public static void main (String[] args) throws Exception {
    SPGSL spgsl = new SPGSL();
    /* Create a problem to optimise with numVariables and 100 Evaluations */
    SampleProblem problem = new SampleProblem(numVariables, 100);
    spgsl.findMinimum(problem); // Starting the routine

    /* Give the best results of the optimisation found at the end of the the routine */
    System.out.println("\n\n The best solution is " +
        df.format(problem.bestPoint.x[0]) + "\n = " +
        df.format(problem.bestPoint.y) + "\n");
}

//----------------------------------------------------------