Lab Assignment 8
Stochastic Techniques in Global Optimization

ACS II

Spring 2020

Assigned: February 27th, 2020

Due: March 5th, 2020

Introduction

When considering global optimization problems, it is generally agreed that there is no 'gold-standard' algorithm, with different algorithm types (line search, stochastic) performing variably better or worse than the other depending on the specific problem considered. It is thus useful to have first-hand experience implementing multiple families of global optimization techniques. In the previous lab assignment, you implemented a variety of line-search techniques to optimize a global function. In this lab, you will implement the Controlled Random Search (CRS) algorithm as developed by Price (1977). A key feature of this stochastic method is that it does not rely on information about the gradient or hessian of the objective function. It is likewise not as susceptible to converging on local minima/maxima as Newton's method (or other gradient-based methods), and it is in general more capable of handling a noisy objective function. Once you have mastered this algorithm in serial, you will then exploit the data parallelism inherent in the method, and develop a modifed version of the code which utilizes message-passing-interface (MPI).

Controlled Random Search Algorithm

The CRS algorithm originally developed by Price is an effective algorithm for solving a global optimization problem with or without constraints, as long as the function evaluation is not prohibitively expensive. The key points of the algorithm are highlighted here for convenience. Given a function to minimize $f(\mathbf{x})$, with $\mathbf{x} \in \mathbb{R}^n$, an initial search space $V \in \mathbb{R}^n$ is constructed by randomly generating N points, referred to as the cloud (these points must lie in the feasible region, of course.) The function is evaluated at each point in

the cloud and stored. At each iteration a new trial point P is selected randomly by a subset of the data in the cloud. Then the function is evaluated at this new point to obtain f_P, and compared with the current point M with the maximal value. If f_P is less than f_M, then P is placed in the cloud and M is removed. The points will tend to cluster around minima, and as the number of points N in the cloud increases, so does the probablity that the global minimum will be among them.

The necessary cloud size N for convergence on the global minimum is related to the number of parameters n, with $N >> n$. A suggestion posed by Price is $N \sim 15n$. In our lab, we will make N much larger though, because we want to intentionally slow down the code to elucidate the benefits of parallelization (since our function f is extremely cheap to evaluate, the code would otherwise be hampered by parallel communication overhead.)

What has yet to be discussed is how the trial points are generated at each iteration. There are a variety of modifications to the CRS algorithm which differ mainly in this respect. In our code, we will do so by generating a subset of $n+1$ points from the cloud, $R_1, \ldots, R_{n+1}$, which serve as the vertices of a n-simplex. Then the centroid, G, of the points $R_1, \ldots, R_n$ is computed, and the point R_{n+1} is projected across it as

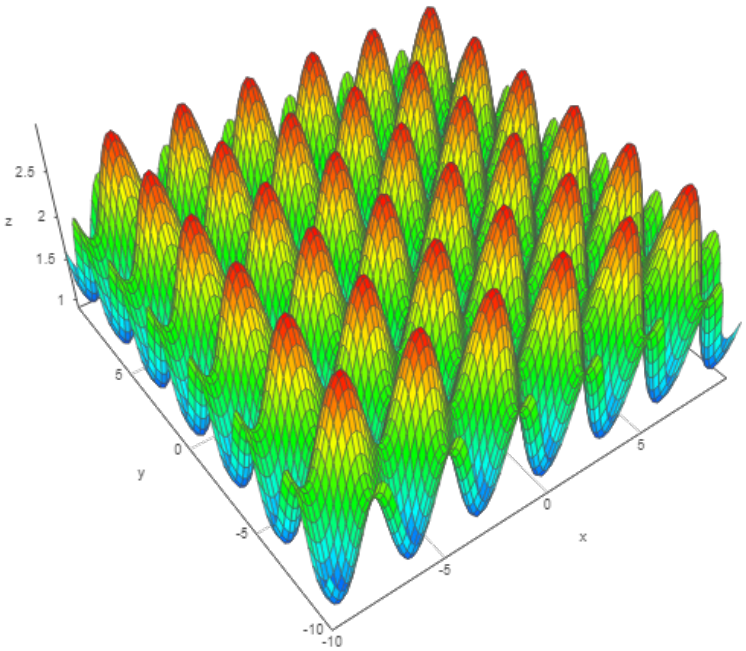
$$\overline{P} = 2\overline{G} - \overline{R}_{n+1},$$

where $\overline{P}, \overline{G}$, and $\overline{R}_{n+1}$ denote the position vectors of the points. While the ordering of the points R_i is generally arbitrary, in practice it is usually beneficial to set R_{n+1} to be the vertex with the smallest function value in the simplex. This process is repeated until a portion of the cloud convergences sufficiently on a global minimum in the sample space (or until a maximum number of iterations are reached.)

Deliverable

Implement Price's CRS algorithm in Fortran, C, or C++. Then, apply it the following function from Price's paper (example 3).

$$f(x, y) = 1 + \sin^2(x) + \sin^2(y) - 0.1e^{-x^2-y^2}.$$



This is a multimodal function, with infinitely many local maxima and minima. There is however a unique global minimum at $(0, 0)$ having a minimum of 0.9.

Let your initial search domain be $(-20 < x < 20, -20 < y < 20)$, and your initial number of cloud points $N = 1000$. Run for a maximum of 10000 iterations (be patient). Compare your results for this optimization problem to those given in the paper. Show the distribution of the points in the cloud on top of a plot of the objective function at the iterations $1000, 5000, 7500$ and 10000.

Parallel Controlled Random Search Algorithm

We can imagine a parallel version of the previous algorithm, where each of the N_p processors simultaneously compute their own trial point, and may potentially update the cloud. This type of scheme makes use of data-parallelism. However a key concern is the sending and recieving of information in the form of updates to the cloud by each of the processors. If the information is not synchronized, it could lead to issues. A common issue faced when designing parallel algorithms is load-balancing. Proper load-balancing ensures that processors are not

idle, waiting on information from one or more processes.

With this scheme we will utilize the same mechanics as the serial code, but each processor will now generate roughly $\frac{1}{N_p}$ of the random points which make up the cloud. This also means that each processor's data set will have its own maximum. Thus when a processor computes a new trial point, it compares the function value at that point with the point corresponding to its own current maximum. Finally, if a trial point is accepted, that processor must broadcast the updated data to the other processors.

Pseudocode for this parallel implementation is supplied below, where a list of essential variables is supplied in comments.

```
// Ndim is the dimensionality in space.
// Nproc is the number of processes
// Ncloud is the cloud size
// X is a (Ncloud, Ndim) matrix of trial solutions
// F is a (Ncloud,1) vector of corresponding cost functions
// X_c is a (Ndim,1) vector corresponding to the centroid
// F_c (a scalar) is the current centroid
// X_m, F_m correspond to the current max point
// X_p, F_p is the trial point selected for update by process p
// RNG_p is a RNG sequence unique for process p

cloud_init()
(X_p,0, F_p,0) = selectPthFromTop( X, F, X_c, Nproc )
while ( .not.solutionFound( X, F ) )
    improvementFound = .false.
    do while ( .not.improvementFound )
          X_p,n+1 = RNG_p( X_c, X_p,n )             // generates new trial point
          F_p,n+1 = *costFunction* ( X_p,n+1 )      // evaluates function at point
          improvementFound = F_p,n+1 <= F_m         // check for improvement
    end do
    receive_cloud_updates ( X, F )
    cloud_update ( X,  F,  X_p,0,  X_p,n+1, F_p,n+1 )
    output_cloud_update
    send_cloud_updates( X, F )
    X_p,0 = X_p,n
end do
```

Deliverable

You are to code the parallel algorithm to solve the same global optimization problem as before. You must utilize MPI send and receive commands in your code. Specifically, the following tasks are required

- Compare how the minimization process differs with the serial case by graphing the difference between the maximum and minimum points in the cloud, as well as just the minimum point in the cloud for $N_p = 1, 2, 4, 8$.
- Using the same number of cloud points as in the serial case, test your code with $N_p = 1, 2, 4, 8$ processes and compare the runtime with the serial case. Note that to obtain a good statistic for each processor measurement, you should take the average of at least three tests.
- After obtaining the runtime measurements, compute the total overhead as

  $$T_O = N_p T_{N_p} - T_1,$$

  and plot it for each value of N_p. Be sure to average your measurements.
- Also compute the speedup as

  $$T_S = \frac{T_1}{T_{N_p}},$$

  and plot it for each value of N_p. Be sure to average your measurements.

Submission and Grading

To get credit for this assignment, you must submit the following information to the lab instructor by 11:59pm, March 5th, 2020:

- Your code.
- Your report as a single file in pdf format, including your plots, results from your work and relevant discussion of your observations, results, and conclusions.

This information must be received by 11:59pm, March 5th, 2020. Upload the required documents to Canvas. As stated in the course syllabus, late assignment submissions will be subject to a 10% point penalty per 24 hours past the due date at time of submission, to a maximum reduction of 50%, according to the formula:

*[final score] = [raw score] - min(0.5, 0.1 * [# of days past due]) * [maximum score]*