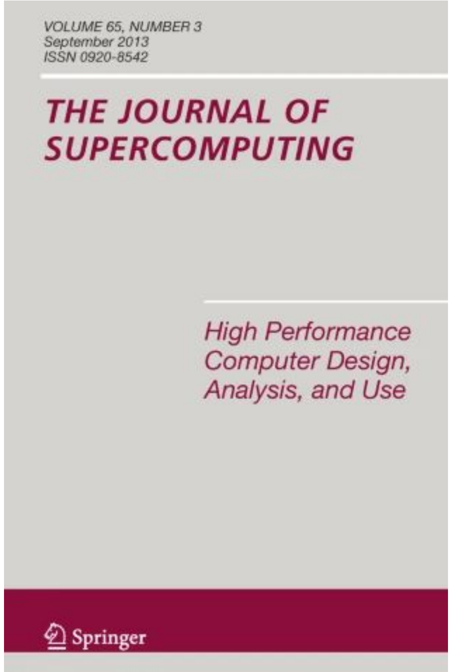
CUDA-based PRISM REFRACTION SEARCH

A novel physics-based metaheuristic optimization algorithm

U22CS060
Shashank Thakur

Introduction



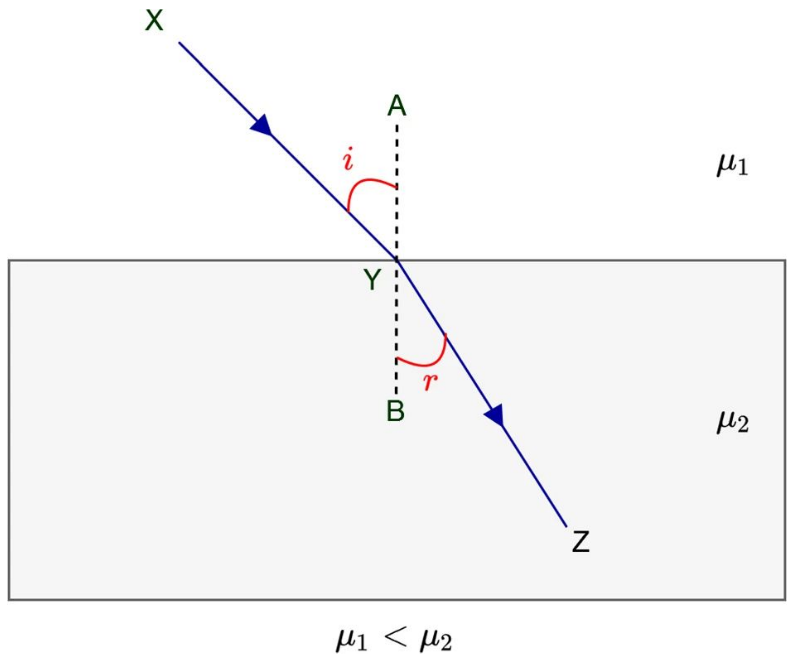
Springer

https://doi.org/10.1007/s11227-023-05790-3

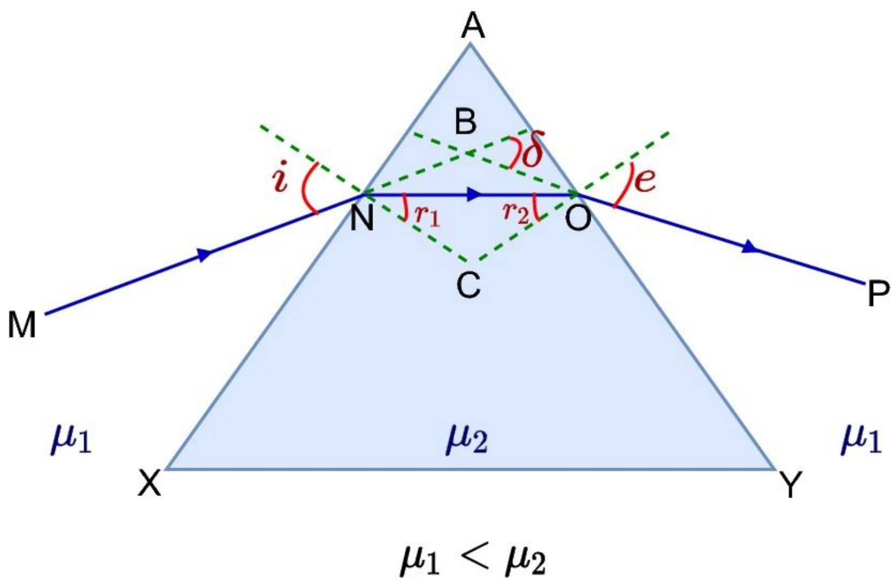
Brief Description

- Physics-based metaheuristic single-solution optimization algorithm
- Models how light interacts with prisms
- Solutions in search space are mapped to light ray angles
- Calculate divergence of those angles as they pass through the prism
- Minimizing the divergence minimizes the function to be optimized

Preliminaries



$$\frac{\sin i}{\sin r} = \frac{\mu_2}{\mu_1} = \mu_{21}$$

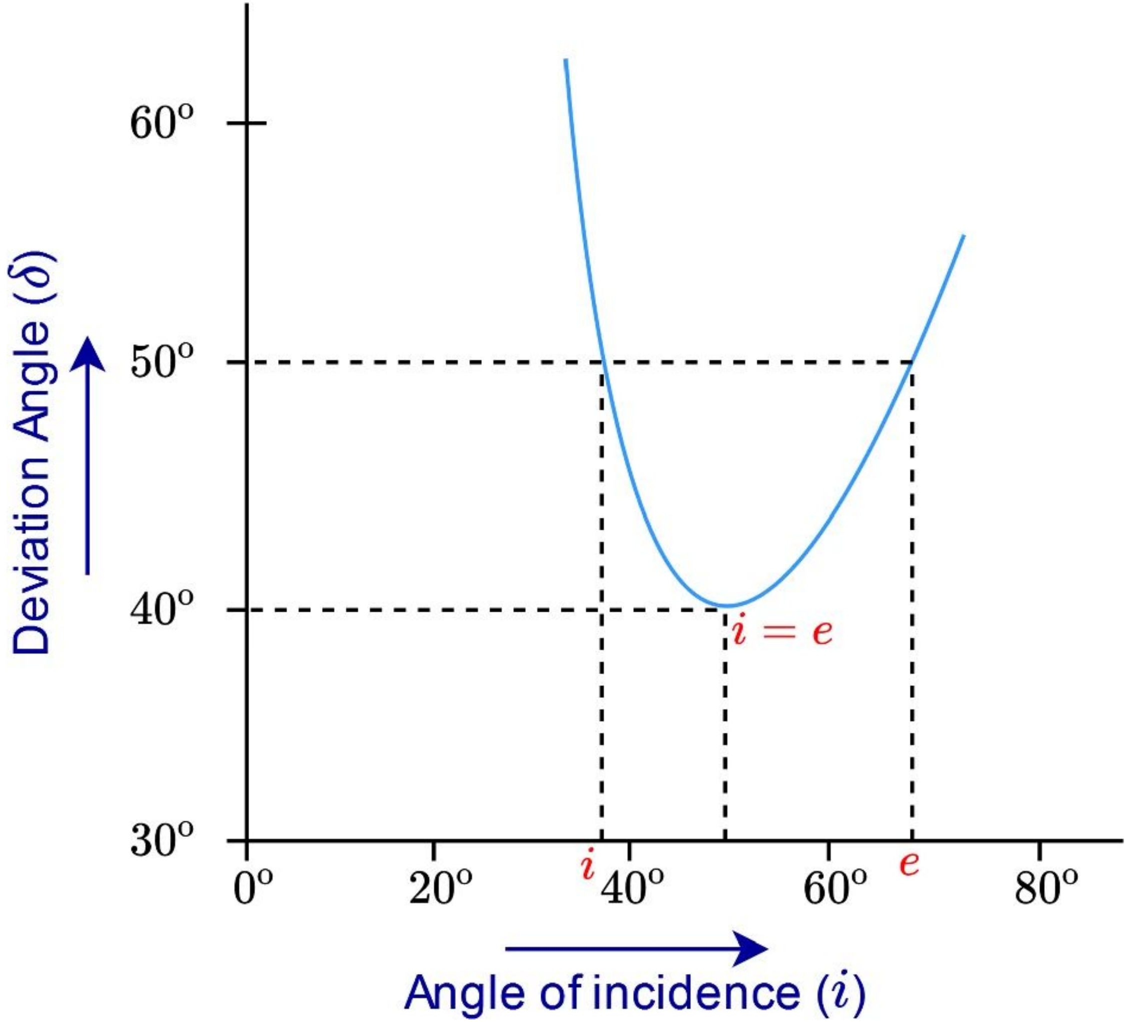


$$r_1 + r_2 = \angle A$$

$$\delta = (i - r_1) + (e - r_2)$$

$$\implies \delta = i + e - \angle A$$

$$
\begin{aligned}
\sin i &= \mu \sin r_1 \\
&= \mu \sin\left(A - r_2\right) \\
&= \mu\left(\sin A \cos r_2 - \cos A \sin r_2\right) \\
&= \mu\left(\sin A \sqrt{1 - \frac{\sin^2 e}{\mu^2}} - \cos A \frac{\sin e}{\mu}\right) \\
&= \sin A \sqrt{\mu^2 - \sin^2 e} - \cos A \sin e \\
i &= \arcsin\left(\sin A \sqrt{\mu^2 - \sin^2 e} - \cos A \sin e\right)
\end{aligned}
$$



60°
50°
40°
30°
0°
20°
40°
60°
80°
$i = e$
i
e
Deviation Angle (δ)
Angle of incidence (i)

$$
\begin{aligned}
\delta_m &= 2i - A \\
\implies i &= \frac{A + \delta_m}{2}
\end{aligned}
$$

$$
\mu_m = \frac{\sin i}{\sin r} = \frac{\sin\left(\frac{A+\delta_m}{2}\right)}{\sin\left(\frac{A}{2}\right)}
$$

Proposed Algorithm

```
Initialize population i_0 denoted by the solution as incident angle, subject to the bounds
defined by prism angle A_0, see Eq.8
for iter = 1 : MaxIters do
    for i = 1 : OneSolution do
        for j = 1 : Dimensions do
            Get fitness calculating the angle of incidence δ_t using Eq.7
            Obtain BestScore
            if(δ_t<BestScore)
            BestScore = fitness
            end if
        end for
    end for
    Calculate the refractive index μ_m using the Eq.10
    for i = 1 : OneSolution do
        for j = 1 : Dimensions do
            Update emergent angle E_t using Eq.9
            Defined a random number r1 into [−1, 1]
            Update incident angle i_{t+1} by Eq.11
        end for
    end for
    Update prism angle A_{t+1} by Eq.12
    Update the best solution and position
end for
```

$$\begin{aligned} i_t &= [i_t^1, i_t^2, i_t^3, \ldots, i_t^N] \\ \delta_t &= \mathcal{F}(i_t) \end{aligned} \tag{7}$$

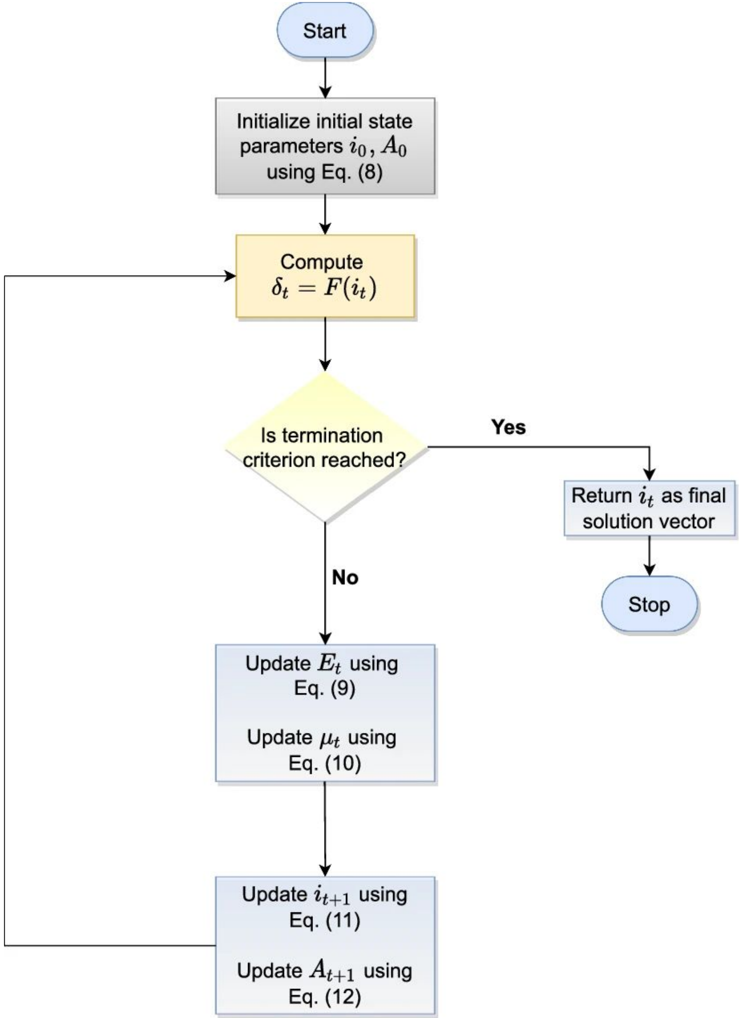
$$\begin{aligned} i_0 &= LB + (UB - LB) \times \mathcal{U}[0, 90] \\ A_0 &= \max_{1 \le j \le N}(LB) + (\min_{1 \le j \le N}(UB) - \max_{1 \le j \le N}(LB)) \times \mathcal{U}[15, 90] \end{aligned} \tag{8}$$

$$E_t = \delta_t - i_t + A_t \tag{9}$$

$$\mu_t = \frac{\sin(\frac{A_t + \delta_t}{2})}{\sin(\frac{A_t}{2})} \tag{10}$$

$$i_{t+1} = \sin^{-1}\left(-\sin E_t \cos A_t + r \times \sin A_t \times \sqrt{\mu_t^2 - \sin^2 E_t}\right) \tag{11}$$

$$A_{t+1} = A_t \times \exp\left(\frac{-\alpha \times t}{MaxIter}\right) \tag{12}$$

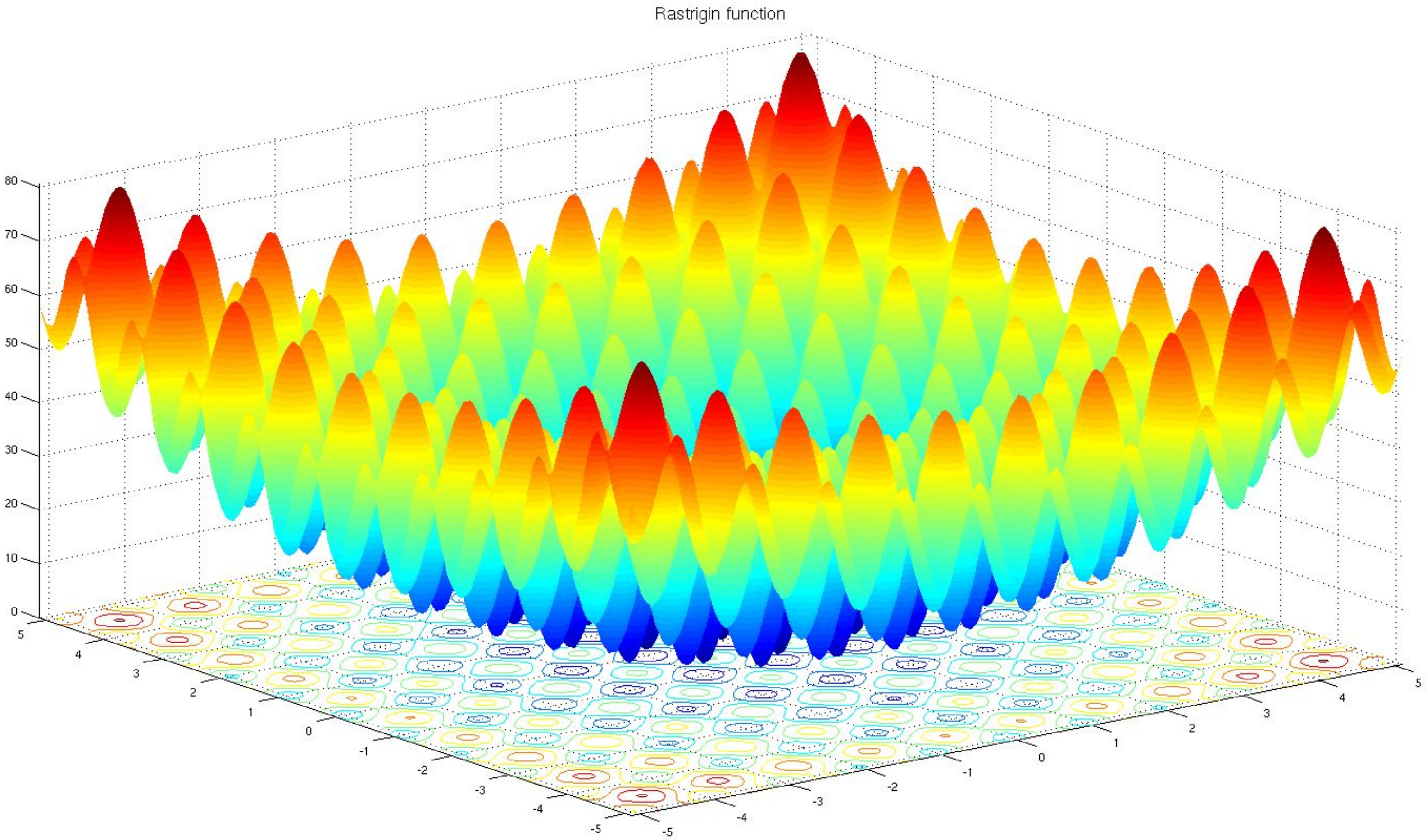

Start
Initialize initial state parameters i_0, A_0 using Eq. (8)
Compute $\delta_t = F(i_t)$
Is termination criterion reached?
Yes
Return i_t as final solution vector
Stop
No
Update E_t using Eq. (9)
Update μ_t using Eq. (10)
Update i_{t+1} using Eq. (11)
Update A_{t+1} using Eq. (12)

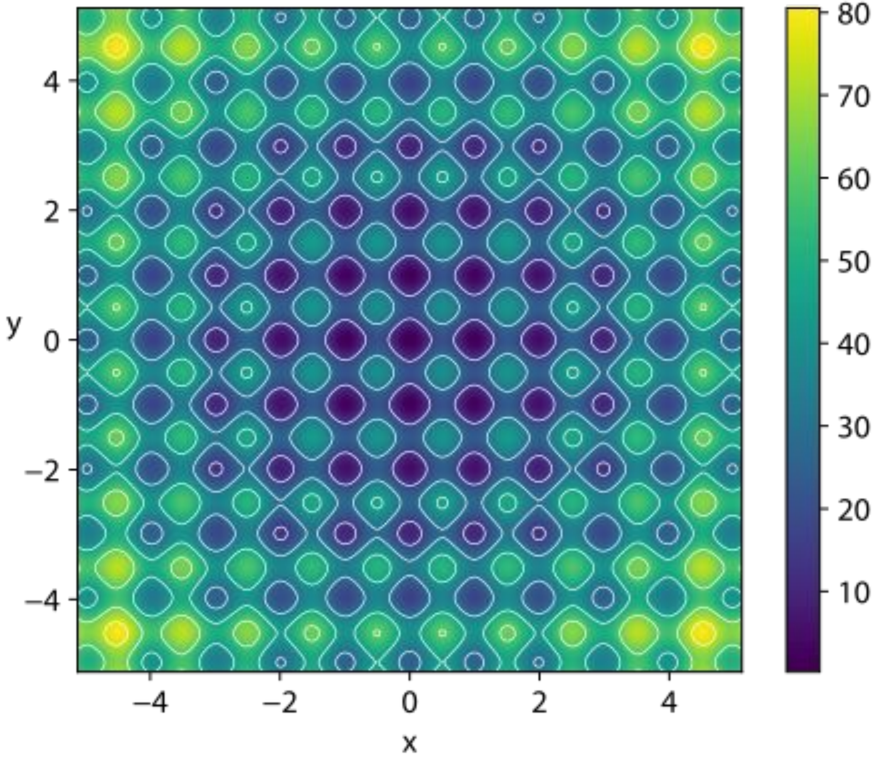
Time Complexity	*O(n x Population x MaxIter)*
Space Complexity	*O(n x Population)*

Benchmark Function

Rastrigin Function

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right]$$


Rastrigin function
80
70
60
50
40
30
20
10
0
5
4
3
2
1
0
-1
-2
-3
-4
-5
-5
-4
-3
-2
-1
0
1
2
3
4
5



80
70
60
50
40
30
20
10
4
2
0
−2
−4
y
−4
−2
0
2
4
x

Performance of CPU-based PRS

```
Parameters:
  Dimension: 4
  Max Iterations: 6000
  Population size: 1000
  Alpha: 0.009000
Best solution:
  0.002293
  0.003727
  -0.004454
  0.007831
Best score: 0.019900
Elapsed time: 53.258303 seconds
```

Accelerating with CUDA

Initialize population i_0 denoted by the solution as incident angle, subject to the bounds defined by prism angle A_0, see Eq.8

for $iter = 1 : MaxIters$ **do**

  for $i = 1 : OneSolution$ **do**

    for $j = 1 : Dimensions$ **do**

      Get fitness calculating the angle of incidence δ_t using Eq.7

      Obtain BestScore

      if $(\delta_t < \text{BestScore})$

      BestScore = fitness

      end if

    end for

  end for

  Calculate the refractive index μ_m using the Eq.10

  for $i = 1 : OneSolution$ **do**

    for $j = 1 : Dimensions$ **do**

      Update emergent angle E_t using Eq.9

      Defined a random number $r1$ into $[-1, 1]$

      Update incident angle i_{t+1} by Eq.11

    end for

  end for

  Update prism angle A_{t+1} by Eq.12

  Update the best solution and position

end for

Initialize population i_0 denoted by the solution as incident angle, subject to the bounds defined by prism angle A_0, see Eq.8

for $iter = 1 : MaxIters$ **do**

for $i = 1 : OneSolution$ **do**

for $j = 1 : Dimensions$ **do**

Get fitness calculating the angle of incidence δ_t using Eq.7

Obtain BestScore

if$(\delta_t <$ BestScore$)$

BestScore = fitness

end if

end for

end for

Calculate the refractive index μ_m using the Eq.10

for $i = 1 : OneSolution$ **do**

for $j = 1 : Dimensions$ **do**

Update emergent angle E_t using Eq.9

Defined a random number $r1$ into $[-1, 1]$

Update incident angle i_{t+1} by Eq.11

end for

end for

Update prism angle A_{t+1} by Eq.12

Update the best solution and position

end for

Initialize population i_0 denoted by the solution as incident angle, subject to the bounds defined by prism angle A_0, see Eq.8

for $iter = 1 : MaxIters$ **do**

    for $i = 1 : OneSolution$ **do**

        for $j = 1 : Dimensions$ **do**

            Get fitness calculating the angle of incidence δ_t using Eq.7

            Obtain BestScore

            if($\delta_t < \text{BestScore}$)

            BestScore = fitness

            end if

        end for

    end for

    Calculate the refractive index μ_m using the Eq.10

    for $i = 1 : OneSolution$ **do**

        for $j = 1 : Dimensions$ **do**

            Update emergent angle E_t using Eq.9

            Defined a random number $r1$ into $[-1, 1]$

            Update incident angle i_{t+1} by Eq.11

        end for

    end for

    Update prism angle A_{t+1} by Eq.12

    Update the best solution and position

end for

Initialize population i_0 denoted by the solution as incident angle, subject to the bounds defined by prism angle A_0, see Eq.8

for $iter = 1 : MaxIters$ **do**

  for $i = 1 : OneSolution$ **do**

    for $j = 1 : Dimensions$ **do**

      Get fitness calculating the angle of incidence δ_t using Eq.7

      Obtain BestScore

      if($\delta_t <$BestScore)

      BestScore = fitness

      end if

    end for

  end for

  Calculate the refractive index μ_m using the Eq.10

  for $i = 1 : OneSolution$ **do**

    for $j = 1 : Dimensions$ **do**

      Update emergent angle E_t using Eq.9

      Defined a random number $r1$ into $[-1, 1]$

      Update incident angle i_{t+1} by Eq.11

    end for

  end for

  Update prism angle A_{t+1} by Eq.12

  Update the best solution and position

end for

```
Kernel to initialize population of N solutions

Initialize prism angle A_0

For t = 1 to MaxIter do

    Kernel to calculate delta for every solution

    Kernel to reduce delta to average delta_t

    Calculate refractive index μ_m:
        μ_m = sin((A_0 + δ_t) / 2 ) / sin(A_0 / 2)   // Refractive index Eq.10

    Kernel to calculate emergent angle           // Eq. 9
        and update incident angles               // Eq. 11

    Update prism angle A_{t+1}:
        A_{t+1} = A_t × ((alpha - t) / MaxIter)      // Prism angle Eq.12

    Kernel to update BestSolution and BestScore

End for

Return BestSolution, BestScore
```

Performance of CUDA-based PRS

```
Number of CUDA devices: 1
Device 0: NVIDIA GeForce MX250
  Total global memory: 4230086656 bytes
  Multiprocessor count: 3
  Max threads per block: 1024
Parameters:
  Dimension: 4
  Max Iterations: 10000
  Population size: 1024
  Alpha: 0.009000
Best solution:
  0.009845
  0.001369
  0.013489
  0.000200
Best score: 0.055679
Elapsed time: 1.451018 seconds
```

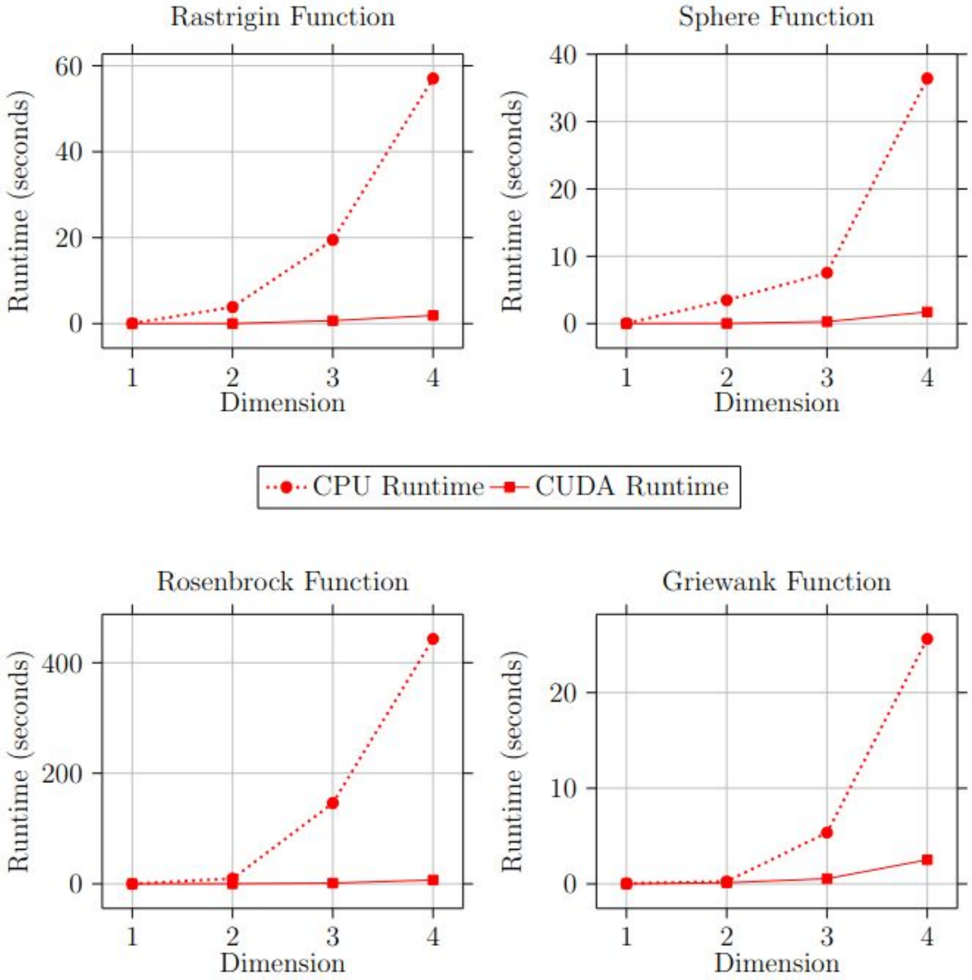
Comparison

CPU

```
Parameters:
  Dimension: 4
  Max Iterations: 6000
  Population size: 1000
  Alpha: 0.009000
Best solution:
  0.002293
  0.003727
  -0.004454
  0.007831
Best score: 0.019900
Elapsed time: 53.258303 seconds
```

CUDA

```
Parameters:
  Dimension: 4
  Max Iterations: 10000
  Population size: 1024
  Alpha: 0.009000
Best solution:
  0.009845
  0.001369
  0.013489
  0.000200
Best score: 0.055679
Elapsed time: 1.451018 seconds
```



Rastrigin Function
Sphere Function
Rosenbrock Function
Griewank Function
Runtime (seconds)
Dimension
CPU Runtime
CUDA Runtime

Conclusion

Comparison

- CUDA PRS performed significantly faster while processing a higher number of iterations (~40 x speedup)
- CUDA processed every solution and (solution, component) pair independently

Algorithm

- The algorithm is mainly an *exploitative* algorithm
- Most of the *exploration* is being done by generating uniformly generated random angles, spread out evenly in the search space
- This algorithm should be used in combination with other algorithms that are good at *exploring* the search to narrow down a possible solution region
- PRS *exploits* the region to find optima

Limitations of my Implementation

- CUDA *cuRAND* random number generator
- Some random numbers are generated in a very small neighborhood, creating a pattern of similarity
- Exploration aspect of this algorithm is hurt
- *XORWOW* and *Philox* states for generating random numbers
- Flattening a 2D array to a 1D array to allocate memory on the GPU
- Memory coalescence for faster memory access

END