COMS BC 3159 - S24: Problem Set 2

Introduction: The following exercises will continue to explore GPU programming and begin to explore numerical optimization.

Finally, we'd like to remind you that all work should be yours and yours alone. This being said, in addition to being able to ask questions at office hours, you are allowed to discuss questions with fellow classmates, provided 1) you note the people with whom you collaborated, and 2) you **DO NOT** copy any answers. Please write up the solutions to all problems independently.

Without further ado, let's jump right in!

Collaborators:
AI Tool Disclosure:

Problem 1 (4 Points):

For the following questions please indicate if the statement is true or false and then explain your answer in 1-3 sentences.

(a) Gradient descent will always find a global minima.

(b) The critical points of a function are where the first derivative (or gradient) of that function is equal to zero. These points are guaranteed to be local minima or maxima.

(c) If a function is convex then the line segment connecting two points on the graph of f always lies above the graph of f.

(d) Practical gradient descent methods often do not apply the full gradient update at each iteration.

Solution 1:

(a) ☐ True ☐ False

(b) ☐ True ☐ False

(c) ☐ True ☐ False

(d) ☐ True ☐ False

Problem 2 (8 Points):

For this problem we are going to work with the function:

$$y(x) = \frac{3}{2e}(ex - e)^2 - e^x$$

(a) (3 points) Compute a quadratic approximation $\tilde{y}(x)$ of the function at the point $x = 1$ through the use of a Taylor expansion. Please show your work. You do not need to simplify your answer completely.

(b) (2 points) What are the critical point(s) of the approximation you computed in part a? Please show your work.

(c) (1 point) What are the critical point(s) of the original function? Note: You do not need to solve this by hand but explain what you did.

(d) (2 points) Are the critical points the same or different? Why or why not? Please explain in 1-3 sentences.

Solution 2:

(a)

(b)

(c)

(d)

Problem 3 (12 Points):

For this problem we are going to work with the function:

$$y(x) = x^4 - 3x^2 + 2x$$

Note: You do not need to solve this by hand but explain what you did in each step (do not just write down the final answer).

(a) (2 points) What are the critical points of this function?

(b) (2 points) Compute one step of gradient descent starting from $x = -2$ with $\alpha = 0.025$. Where do you end up?

(c) (1 point) Based on that step do you think $\alpha = 0.025$ was a good choice? If you were to do it again would you leave it the same? Increase it? Decrease it?

(d) (2 points) Starting at the point computed by part b, take one more step of gradient descent this time using $\alpha = 0.1$. Where do you end up?

(e) (1 point) Based on that step do you think $\alpha = 0.1$ was a good choice? If you were to do it again would you leave it the same? Increase it? Decrease it?

(f) (3 points) Now assume you are starting again from $x = -2$ and now doing gradient descent with a backtracking line search starting with $\alpha = 1$ and updating $\alpha = \alpha/2$ on every backtracking step. With what value of α will the *first* step occur under the cost function $J(x) = (x - nearest\_local\_min)^2$ where $nearest\_local\_min$ is (one of) your value(s) computed in part a? Where will that take you?

(g) (1 point) Based on everything you have explored in this problem, do you think that resulting α from part f was a good choice? If you were to do it again would you leave it the same? Increase it? Decrease it?

Solution 3:

(a)

(b)

(c)

(d)

(e)

(f)

(g)

Problem 4 (8 Points)

Below you will find multiple different examples of GPU kernels designed to add one to the input array in parallel. For this question please assume that the input array is of length 128 and the host code was done correctly. That is, `N=128` and `d_data` was correctly allocated on the GPU to be that length. For each kernel indicate ALL of the kernel launch configurations that will produce the correct output. If none are correct please specify what configuration would work! Launch configuration options are:

i `increment_kernel<<<1,1,0,0>>>(d_data,128);`

ii `increment_kernel<<<1,128,0,0>>>(d_data,128);`

iii `increment_kernel<<<1,256,128*sizeof(int),0>>>(d_data,128);`

iv
```
dim3 tDims(128,128);
increment_kernel<<<1,tDims,0,0>>>(d_data,128);
```

v
```
dim3 bDims(128,128); tDims(128,128);
increment_kernel<<<bDims,tDims,0,0>>>(d_data,128);
```

vi None of the above (please specify the correct configuration)

Kernel (a)

```
1 __global__
2 void increment_kernel(int *d_data, int N) {
3     if (threadIdx.x < N) {
4         d_data[threadIdx.x] += 1;
5     }
6 }
```

Kernel (b)

```
1 __global__
2 void increment_kernel(int *d_data, int N) {
3     for (int i = threadIdx.x; i < N; i += blockDim.x) {
4         d_data[i] += 1;
5     }
6 }
```

Kernel (c)

```
1  __global__
2  void increment_kernel(int *d_data, int N) {
3      d_data[threadIdx.x] += 1;
4  }
```

Kernel (d)

```
1  __global__
2  void increment_kernel(int *d_data, int N) {
3    int thread_rank = threadIdx.x + blockDim.x*threadIdx.y;
4    int threads_per_block = blockDim.x*blockDim.y;
5    int block_rank = blockIdx.x + gridDim.x*blockIdx.y;
6    int blocks_per_grid = gridDim.x*gridDim.y;
7    int grid_rank = thread_rank + block_rank*threads_per_block;
8    int threads_per_grid = threads_per_block*blocks_per_grid;
9    for (int i = grid_rank; i < N; i += threads_per_grid) {
10     d_data[i] += 1;
11   }
12 }
```

Solution 4:

(a)

(b)

(c)

(d)