

NAG Numerical Functions for GPUs Function Document

nag_gpu_sobol_exp

1 Purpose

nag_gpu_sobol_exp generates multidimensional quasi-random sequences from an exponential distribution with mean 1.0

The initialization function `nag_gpu_sobol_init` must be called prior to the first call to **nag_gpu_sobol_exp**. Successive calls to **nag_gpu_sobol_exp** will generate successive values along the sequence. Once all desired values have been obtained, the function `nag_gpu_sobol_cleanup` must be called to free allocated system resources.

The dimension of the Sobol sequence is given by the `dim` parameter in `nag_gpu_sobol_init`. All references to `dim` below refer to that value.

2 Specification

```
#include <nag_gpu.h>
```

```
void nag_gpu_sobol_exp (int npts, float *d_P, Nag_Gpu_Comm *comm)
```

```
void nag_gpu_sobol_exp (int npts, double *d_P, Nag_Gpu_Comm *comm)
```

3 References

None.

4 Arguments

1: **npts** – int *Input*

On entry: the number of quasi-random points required.

Constraint: $1 \leq \text{npts}$.

2: **d_p[npts × dim]** – float * *Output*

3: **d_p[npts × dim]** – double * *Output*

This overloaded parameter can take type double or float.

The array must be allocated in the *device memory space*.

On exit: the `npts` quasi-random values of dimension `dim` from an exponential distribution. For the n th Sobol point $x_n = (x_n^1, x_n^2, \dots, x_n^{\text{dim}})$ with $0 \leq n < \text{npts}$, the d th dimension x_n^d will be stored at location `d_P[(d-1)*npts+n]`. In other words, the first `npts` values correspond to dimension 1, the second `npts` to dimension 2, and so on.

4: **comm** – Nag_Gpu_Comm * *Communication Data*

`Nag_Gpu_Comm` is a NAG defined type which holds state and communication information and must not be modified in any way.

The structure initialized by a previous call to `nag_gpu_sobol_init`.

5 Error Indicators and Warnings

No argument constraint checking is carried out by this function. You can insert a call to function `nag_gpu_utilCheckMsg` (contained in the header file `nag_gpu.h`) following the call to **nag_gpu_sobol_exp** to check for the last CUDA error message, if any, generated during execution.

6 Example

This example prints the first 64 quasi-random numbers of dimension 5 from an exponential distribution generated by `nag_gpu_sobol_exp`. The initialization function `nag_gpu_sobol_init` is called first to prepare the Sobol generator.

6.1 Program Text

```

/* nag_gpu_sobol_test
 *
 *
 * Copyright 2010, Numerical Algorithms Group Ltd, Oxford, UK.
 *
 * Version 0.3, 2010.
 *
 */

////////////////////////////////////
// pre-processor includes
////////////////////////////////////
#include <stdio.h>

////////////////////////////////////
// include libraries
////////////////////////////////////
#include <nag_gpu.h>

////////////////////////////////////
// define precision
////////////////////////////////////
#define PRS float

////////////////////////////////////
// Main program
////////////////////////////////////

int main(int argc, char **argv)
{
    // fopen("nag_gpu_sobol_exponential_example.results.txt","w");
    FILE *fpout = stdout;

    // Max dims supported by Sobol generator is 50,000
    const int dim = 40;
    // Generate batchSize numbers at a time on the GPU. For large sample
    // sizes this can avoid out-of-memory errors
    const int batchSize = 50;

    // Total number of batches to generate
    const int nBatches = 10;

    PRS
    // Host (PC) storage for batch-generated GPU Sobol numbers
    *h_GpuBatchNums = 0,
    // Device (GPU) storage for batch-generated Sobol numbers
    *d_GpuBatchNums = 0;

    Nag_Gpu_Comm
    // Communication structure pointer for GPU generator
    comm;

    // Scrambling vector for GPU Sobol
    unsigned int *v = 0;
    // Offset into Sobol sequence
    const int offset = 0;

```

```

// Number of dimensions to print: must be less than dim
const int printDim = 5;
// Number of Sobol values to print
int nPrint = 64;
// Make sure we have enough numbers
nPrint =
    (nPrint > batchSize*nBatches
     ? batchSize*nBatches : nPrint);
// Storage for numbers we'll print
PRS *h_print = 0;
// Offset into h_print at which to store GPU numbers
int printOffset = 0;

// Greeting
fprintf(fpout, "\n\n NAG GPU Sobol Example Program: exponential\n\n");

// Set device with highest Gflops/s
cudaSetDevice( nag_gpu_GetMaxGflopsDeviceId() );

// Allocate memory
if (!(h_GpuBatchNums = (PRS*)malloc(sizeof(PRS)*dim*batchSize)) ) {
    fprintf(stderr, "h_GpuBatchNums allocation failure\n");
    goto END;
}
if (!(cudaSuccess == cudaMalloc((void **)&d_GpuBatchNums,
    sizeof(PRS)*dim*batchSize))) {
    fprintf(stderr, "d_GpuBatchNums allocation failure\n");
    goto END;
}
if (!(v = (unsigned int*)malloc(sizeof(unsigned int)*dim)) ) {
    fprintf(stderr, "v allocation failure\n");
    goto END;
}
if (!(h_print = (PRS*)malloc(sizeof(PRS)*printDim*nPrint)) ) {
    fprintf(stderr, "h_print allocation failure\n");
    goto END;
}

// Set scrambling vector to 0 - don't want scrambling
for (int i=0; i < dim; i++) {
    v[i] = 0;
}

// Initialise GPU Sobol generator
nag_gpu_sobol_init(dim, offset, v, &comm);

for(int batchCount = 0; batchCount < nBatches; batchCount++) {
    // Launch GPU computation asynchronously
    nag_gpu_sobol_exp(batchSize, d_GpuBatchNums, &comm);

    /*
     * One can now launch other kernels to operate on the Sobol numbers,
     * or copy the numbers to the host and operate on them there.
     * Here we simply copy them to the host in order to print them
     */

    cudaMemcpy(h_GpuBatchNums, d_GpuBatchNums,
        sizeof(PRS)*dim*batchSize, cudaMemcpyDeviceToHost);

    // Copy GPU values to print storage
    for (int i = 0; i < batchSize; i++) {
        if (i+printOffset < nPrint) {
            for (int j = 0; j < printDim; j++) {
                // We transpose the GPU data for easier display
                h_print[(printOffset+i)*printDim + j] =
                    h_GpuBatchNums[i+j*batchSize];
            }
        }
    } // END print storage loop
}

```

```

    printOffset += batchSize;
} // END batching for loop

// Print Sobol numbers
fprintf(fpout,
        "\nThe first 64 GPU numbers from dimensions 1 to 5:\n");
fprintf(fpout, "    n ");
for(int d = 1; d<=printDim; d++) fprintf(fpout, "    dim%-2d ", d);
for(int i = 0; i < nPrint; i++) {
    fprintf(fpout, "\n%5d", i+1);
    for(int d = 0; d < printDim; d++)
        fprintf(fpout, " %10.4f ", h_print[i*printDim+d]);
}
fprintf(fpout, "\n\n");

END:

// Release GPU and CPU memory
nag_gpu_sobol_cleanup(&comm);

if (d_GpuBatchNums)
    nag_gpu_utilSafeCall( cudaFree(d_GpuBatchNums) );

if (h_GpuBatchNums)
    free(h_GpuBatchNums);

if (v)
    free(v);

if (h_print)
    free(h_print);

fflush(stdout);
fflush(stderr);

if (fpout!=stdout)
    fclose(fpout);

cudaThreadExit();
return 0;
}

```

6.2 Program Data

None.

6.3 Program Results

NAG GPU Sobol Example Program: exponential

The first 64 GPU numbers from dimensions 1 to 5:

n	dim1	dim2	dim3	dim4	dim5
1	22.1807	22.1807	22.1807	22.1807	22.1807
2	0.6931	0.6931	0.6931	0.6931	0.6931
3	0.2877	1.3863	1.3863	1.3863	0.2877
4	1.3863	0.2877	0.2877	0.2877	1.3863
5	0.9808	0.9808	0.4700	0.1335	0.9808
6	0.1335	0.1335	2.0794	0.9808	0.1335
7	0.4700	2.0794	0.1335	0.4700	0.4700
8	2.0794	0.4700	0.9808	2.0794	2.0794
9	1.6740	1.1632	0.0645	0.8267	0.5754
10	0.3747	0.2076	0.8267	0.0645	2.7726
11	0.0645	2.7726	0.3747	1.6740	1.1632

12	0.8267	0.5754	1.6740	0.3747	0.2076
13	1.1632	1.6740	1.1632	0.5754	0.0645
14	0.2076	0.3747	0.2076	2.7726	0.8267
15	0.5754	0.8267	2.7726	0.2076	1.6740
16	2.7726	0.0645	0.5754	1.1632	0.3747
17	2.3671	0.7577	0.7577	0.4212	1.2685
18	0.5213	0.0317	0.0317	1.8563	0.2469
19	0.1699	1.5198	1.5198	0.0984	0.6325
20	1.0678	0.3302	0.3302	0.9008	3.4657
21	0.7577	2.3671	0.1699	1.2685	1.8563
22	0.0317	0.5213	1.0678	0.2469	0.4212
23	0.3302	1.0678	0.5213	3.4657	0.0984
24	1.5198	0.1699	2.3671	0.6325	0.9008
25	1.8563	1.8563	0.6325	0.1699	0.1699
26	0.4212	0.4212	3.4657	1.0678	1.0678
27	0.0984	0.9008	0.2469	0.5213	2.3671
28	0.9008	0.0984	1.2685	2.3671	0.5213
29	1.2685	1.2685	1.8563	1.5198	0.3302
30	0.2469	0.2469	0.4212	0.3302	1.5198
31	0.6325	3.4657	0.9008	0.7577	0.7577
32	3.4657	0.6325	0.0984	0.0317	0.0317
33	3.0603	1.3257	0.3522	0.6035	1.9617
34	0.6035	0.2671	1.5939	3.0603	0.4453
35	0.2271	4.1589	0.0480	0.2271	0.1158
36	1.2144	0.6624	0.7916	1.2144	0.9400
37	0.8630	1.9617	2.5494	0.8630	1.3257
38	0.0813	0.4453	0.5480	0.0813	0.2671
39	0.3977	0.9400	1.1144	1.7610	0.6624
40	1.7610	0.1158	0.1886	0.3977	4.1589
41	1.4508	2.5494	1.3257	0.0157	0.3522
42	0.3087	0.5480	0.2671	0.7249	1.5939
43	0.0157	1.1144	4.1589	0.3087	0.7916
44	0.7249	0.1886	0.6624	1.4508	0.0480
45	1.0234	0.7916	0.1158	2.2130	0.1886
46	0.1515	0.0480	0.9400	0.4953	1.1144
47	0.4953	1.5939	0.4453	1.0234	2.5494
48	2.2130	0.3522	1.9617	0.1515	0.5480
49	2.5494	1.4508	0.2271	1.9617	0.8630
50	0.5480	0.3087	1.2144	0.4453	0.0813
51	0.1886	0.7249	0.6035	0.9400	0.3977
52	1.1144	0.0157	3.0603	0.1158	1.7610
53	0.7916	1.0234	0.8630	0.2671	3.0603
54	0.0480	0.1515	0.0813	1.3257	0.6035
55	0.3522	2.2130	1.7610	0.6624	0.2271
56	1.5939	0.4953	0.3977	4.1589	1.2144
57	1.9617	0.8630	1.4508	1.1144	0.0157
58	0.4453	0.0813	0.3087	0.1886	0.7249
59	0.1158	1.7610	0.7249	2.5494	1.4508
60	0.9400	0.3977	0.0157	0.5480	0.3087
61	1.3257	3.0603	0.4953	0.3522	0.4953
62	0.2671	0.6035	2.2130	1.5939	2.2130
63	0.6624	1.2144	0.1515	0.0480	1.0234
64	4.1589	0.2271	1.0234	0.7916	0.1515
