New Bounds for Matrix Multiplication: from Alpha to Omega

Table of Contents

General Information

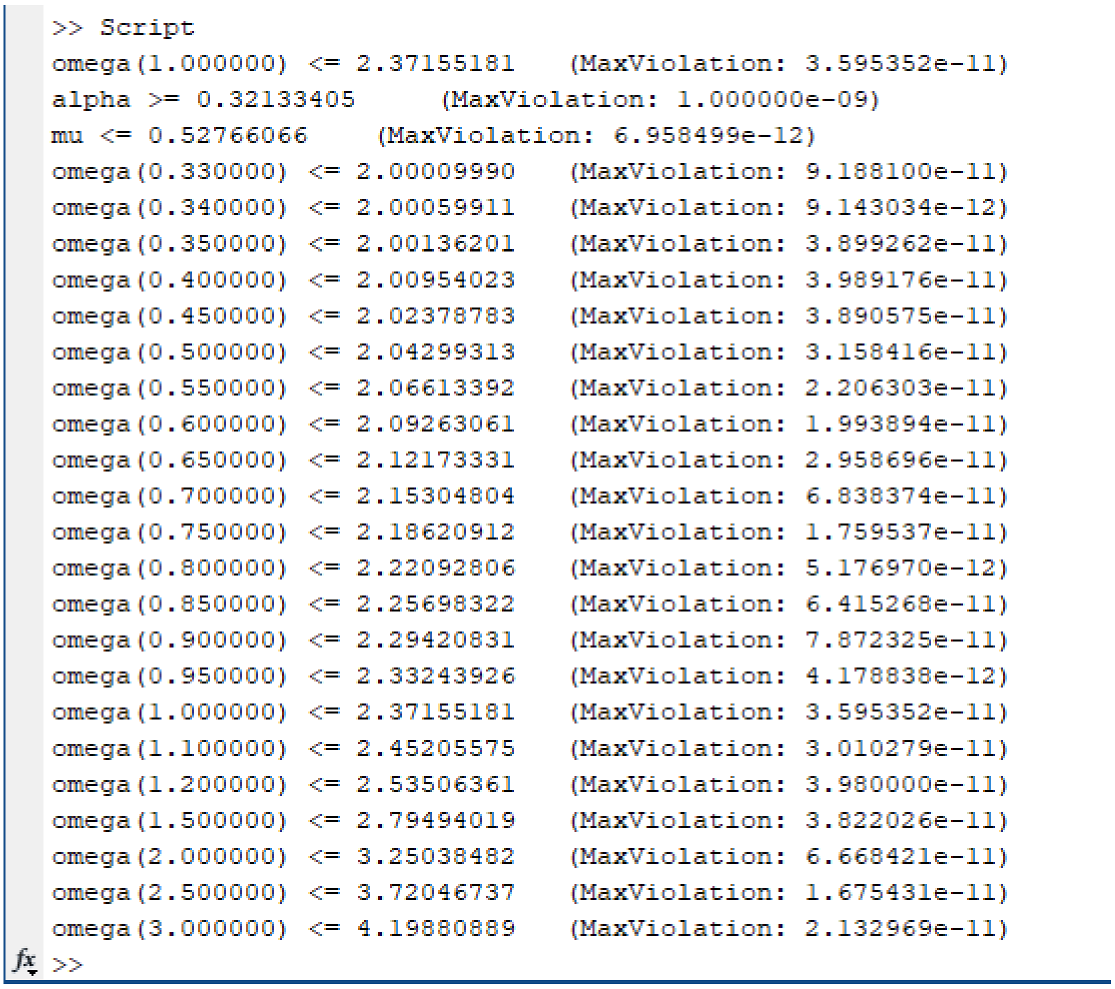
This project is the optimization & verification code for the paper *New Bounds for Matrix Multiplication: from Alpha to Omega*. For a given number $\kappa > 0$, the program tries to obtain an upper bound on $\omega(\kappa) := \omega(\kappa, 1, 1) = \omega(1, \kappa, 1) = \omega(1, 1, \kappa)$. It also obtains bounds for the *dual matrix multiplication exponent* α as well as the value μ related to the complexity of *unweighted All-Pairs Shortest Paths (undirected APSP)*. Specifically, the following bounds can be obtained or verified:

- $\omega \leq 2.371552$;
- $\alpha \geq 0.321334$;
- $\mu \leq 0.527661$.

Technologies Used

- MATLAB R2022a
- SNOPT Version 7.7
- CVX Version 2.2
- (Optional) Mosek Version >= 9.3.20

Screenshots



```
>> Script
omega(1.000000) <= 2.37155181   (MaxViolation: 3.595352e-11)
alpha >= 0.32133405      (MaxViolation: 1.000000e-09)
mu <= 0.52766066     (MaxViolation: 6.958499e-12)
omega(0.330000) <= 2.00009990   (MaxViolation: 9.188100e-11)
omega(0.340000) <= 2.00059911   (MaxViolation: 9.143034e-12)
omega(0.350000) <= 2.00136201   (MaxViolation: 3.899262e-11)
omega(0.400000) <= 2.00954023   (MaxViolation: 3.989176e-11)
omega(0.450000) <= 2.02378783   (MaxViolation: 3.890575e-11)
omega(0.500000) <= 2.04299313   (MaxViolation: 3.158416e-11)
omega(0.550000) <= 2.06613392   (MaxViolation: 2.206303e-11)
omega(0.600000) <= 2.09263061   (MaxViolation: 1.993894e-11)
omega(0.650000) <= 2.12173331   (MaxViolation: 2.958696e-11)
omega(0.700000) <= 2.15304804   (MaxViolation: 6.838374e-11)
omega(0.750000) <= 2.18620912   (MaxViolation: 1.759537e-11)
omega(0.800000) <= 2.22092806   (MaxViolation: 5.176970e-12)
omega(0.850000) <= 2.25698322   (MaxViolation: 6.415268e-11)
omega(0.900000) <= 2.29420831   (MaxViolation: 7.872325e-11)
omega(0.950000) <= 2.33243926   (MaxViolation: 4.178838e-12)
omega(1.000000) <= 2.37155181   (MaxViolation: 3.595352e-11)
omega(1.100000) <= 2.45205575   (MaxViolation: 3.010279e-11)
omega(1.200000) <= 2.53506361   (MaxViolation: 3.980000e-11)
omega(1.500000) <= 2.79494019   (MaxViolation: 3.822026e-11)
omega(2.000000) <= 3.25038482   (MaxViolation: 6.668421e-11)
omega(2.500000) <= 3.72046737   (MaxViolation: 1.675431e-11)
omega(3.000000) <= 4.19880889   (MaxViolation: 2.132969e-11)
>>
```

Getting Started

- Simply run `Script.m` to verify all provided bounds on $\omega(\kappa)$.
- Use functions `StartFromScratch`, `StartFromAnother`, `StartAlphaFromAnother`, `StartMuFromAnother` (see source files under `src/control`) to start an optimization process. Before running the optimization, the user needs to replace the path to the SNOPT license in `src/utils/InitSNOPT.m`. It is recommended to first read the user guide below.

User Guide

Framework

The core part of this project is to define a system of nonlinear constraints of the parameters, see directory `src/evaluation`. Given these constraints, one can easily verify whether a set of parameters will imply a bound on $\omega(\kappa)$.

Next, `src/autograd/GVar.m` defines a class `GVar` which uses forward propagation to maintain the derivatives of every intermediate value with respect to every optimizable parameter. `ParamManager.m` under the same directory also helps managing all parameters.

Finally, the functions under `src/optimizer` transforms our constraints into desired formats and call SNOPT to solve the optimization problem.

Running an Optimization

The function `StartFromScratch` runs the optimization from the initial point induced by Le Gall's parameters for square matrix multiplication. Currently this function only supports $q = 5$ for the fourth power of the CW tensor, where Le Gall's parameters in this case are stored in `primitive_param_legall.json`, which is exported from another program reproducing Le Gall's results in [Le Gall ISSAC'14]. To use this function, the user needs to first ensure that CVX and SNOPT are properly installed, and replace the path to the SNOPT license in `src/utils/InitSNOPT.m` according to your environment. Then execute

```
AddPath;
StartFromScratch(5, K, 'expname');
```

Here, the first argument represents $q = 5$; the second argument denotes κ where we are solving for an upper bound of $\omega(\kappa)$ (it is recommended to run for $\kappa = 1$ first); the last argument is a string indicating the name of the experiment.

Each optimization run is managed as an *experiment*, which is associated with a name, e.g., `expname`. Then, all data files of this run will be stored in the directory `expname`. They include:

- `log.txt`: the log file of this run. It records the history of finding new best solutions and other similar events.
- `manual_stop.txt`: contains a single 0 or 1. Once the user writes 1 into this file, SNOPT will quit after the current major iteration. This file is automatically created (with a 0 in it) at the beginning of each run.
- `best.mat`: the best known solution so far. Every time when SNOPT evaluates some solution, we automatically check if the solution is *feasible* (usually this means the maximum violation of constraints does not exceed 10^{-6}), and the feasible solution with the best objective is stored in `best.mat`.
- `sn_output.out` and `sn_param.mat`: the former is automatically generated by SNOPT and the latter stores the values of parameters when SNOPT exits. They are mainly for debugging use.

By executing the function `StartFromScratch`, MATLAB first calls a procedure to transform the json file into the expected format in our framework. CVX, a convex optimization package, is used in this procedure. Next, SNOPT will try to find a solution that imply an upper bound on $\omega(\kappa)$.

However, SNOPT sometimes quits abnormally due to unexpected numerical difficulties. In this case, we slightly perturb the current best solution randomly and try to restart SNOPT. In addition, once the best known solution is not improved for an hour, which is often the case if the random perturbation is not ideal, we terminate SNOPT and try another perturbation. The program is likely to repeatedly try perturbations until the user gives a manual stop signal in `expname/manual_stop.txt`. The only case that the optimization program exits normally is when SNOPT finds a local minima with high precision, which is unusual in practice because the precision requirement is too high.

After terminating the optimization program, the best known solution with max-violation less than 10^{-6} is stored in `expname/best.mat`. It usually takes few days for an optimization run to complete (i.e., no new improvement is observed after several trials of perturbation).

Start from a Nearby Solution

Compared to running an optimization from Le Gall's initial point, it is often more efficient to start the optimization from a nearby solution. For example, to obtain an upper bound for $\omega(0.95)$, we can start with the parameters for $\omega(1)$, given that we have finished the optimization for $\omega(1)$ from Le Gall's initial point. This procedure is implemented as the function `StartFromAnother`:

```
StartFromAnother(5, new_K, experiment_name, old_K, 'path/to/datafile.mat');
```

It loads a given data file and start the optimization process for a new κ.

For objectives α and μ, we provide similar functions:

```
StartAlphaFromAnother(5, experiment_name, 'path/to/datafile.mat');
StartMuFromAnother(5, experiment_name, 'path/to/datafile.mat');
```

Refining Mode

We implemented a subroutine `SnRefine` that can refine the current solution with a higher precision requirement (10^{-9}). This will not make much change to the low-precision (10^{-6}) solution, but it makes our reported bounds more trustable. After an optimization run is completed, call `SnRefine()` to start SNOPT again in the refining mode. The behavior of the refining mode is almost the same as the normal mode except the precision requirement. Similar to before, the best known solution with max-violation less than 10^{-9} will be stored in `expname/best.mat` (overwriting the best solution of the previous run, so the user may want to backup this file before calling `SnRefine`).

Verifying the Provided Parameters

The authors provide all parameters that imply the reported bounds in the `data` directory. For example, `data/K75_2.18620911.mat` contains the parameters used for the bound $\omega(0.75) \le 2.18620911$ (the substring `K75` in the file name denotes $\kappa = 0.75$). The user can verify a single data file using functions `VerifyOmega`, `VerifyAlpha`, and `VerifyMu` under directory `src/verify`. One can also verify all provided data files using the script `Script.m` on the root of this code repository.

Naming

The variable naming style may differ from the expressions in the paper. For a better understanding of the code, we provide a list of frequently used variable names along with their meanings:

- `shape`: a triple of indices (i, j, k) of a constituent tensor.
- `column`: basically the same as "constituent tensor" or "shape". Consider a triple $X_I Y_J Z_K$ of $(\mathrm{CW}_q^{\otimes 2^{\ell-1}})^{\otimes n}$, if we write I, J, K as three rows of a $3 \times n$ matrix, every column is a shape (I_t, J_t, K_t). The term `column` refers to this shape.
- `hash_penalty_term`: also known as $P_\alpha = \max_{\alpha' \in D'} H(\alpha') - H(\alpha)$, which measures the loss in the number of remaining triples that arises when the selected distribution α does not have the maximum entropy among all distributions with the same marginals.
- `part`: often refers to a *term* in the interface tensor.
- `K`: sometimes refers to κ where we want to obtain a bound for $\omega(\kappa)$.