October 23, 2018 *David Simmons-Duffin*

SDPB 1.0

Contents

1 Introduction

SDPB is an arbitrary-precision semidefinite program solver, specialized for "polynomial matrix programs" (defined below). This document describes SDPB's usage and input/output. Much more detail about its design is given in [1]. The reader is encouraged to look there for a better understanding of SDPB's parameters and internal operation.

1.1 Installation and Requirements

`SDPB` requires

- The Boost C++ Libraries (tested with Boost 1.54).
- The GNU Multiprecision Library.

To install, you must first edit the `Makefile` to define the variables `GMPINCLUDEDIR`, `BOOSTINCLUDEDIR`, and `LIBDIR`. Then type `make` to build the `sdpb` executable.

2 Polynomial Matrix Programs

`SDPB` solves the following type of problem, which we call a *polynomial matrix program* (PMP). Consider a collection of symmetric polynomial matrices

$$M_j^n(x) \quad = \quad \begin{pmatrix} P^n_{j,11}(x) & \dots & P^n_{j,1m_j}(x) \\ \vdots & \ddots & \vdots \\ P^n_{j,m_j1}(x) & \dots & P^n_{j,m_jm_j}(x) \end{pmatrix} \tag{2.1}$$

labeled by $0 \leq n \leq N$ and $1 \leq j \leq J$, where each element $P^n_{j,rs}(x)$ is a polynomial in x. Given $b \in \mathbb{R}^N$, we would like to

$$\begin{array}{ll} \text{maximize} & b_0 + b \cdot y \quad \text{over} \quad y \in \mathbb{R}^N, \\ \text{such that} & M_j^0(x) + \sum_{n=1}^N y_n M_j^n(x) \succeq 0 \quad \text{for all } x \geq 0 \text{ and } 1 \leq j \leq J. \end{array} \tag{2.2}$$

The notation $M \succeq 0$ means "M is positive semidefinite."

3 Input to `SDPB`

`SDPB` takes the following input:

- for each $j = 1, \dots, J$:
  - polynomial matrices $M_j^0(x), \dots, M_j^N(x)$ of maximum degree d_j,
  - bilinear bases $q_m^{(j)}(x)$ ($m = 0, \dots, \lfloor d_j/2 \rfloor$),
  - sample points $x_k^{(j)}$ ($k = 0, \dots, d_j$),
  - sample scalings $s_k^{(j)}$ ($k = 0, \dots, d_j$),
- an objective function $b_0 \in \mathbb{R}$ and $b \in \mathbb{R}^N$.

A bilinear basis is a collection of polynomials $q_m^{(j)}(x)$ such that $\deg q_m^{(j)} = m$, for example monomials $q_m^{(j)}(x) = x^m$. (A better choice for numerical stability is usually orthogonal polynomials on the positive real line.) The sample points and sample scalings determine how the PMP is represented internally as an SDP. In principle, they do not affect the solution of the PMP, but in practice they can affect numerical stability. The constant b_0 is completely irrelevant to the solution algorithm, but is included for convenience. See [1] for details.

3.1 Input Format

`SDPB` reads the data above in the following XML format.

Listing 1: XML input format for `SDPB`

input to SDPB $\equiv$
<sdp>
$\langle$*xml for objective*$\rangle$
$\langle$*xml for polynomial vector matrices*$\rangle$
</sdp>

xml for objective $\equiv$
<objective>
<elt>b_0</elt>
...
<elt>b_N</elt>
</objective>

xml for polynomial vector matrices $\equiv$
<polynomialVectorMatrices>
$\langle$*xml for polynomial vector matrix* $M_1^n(x)\rangle$
...
$\langle$*xml for polynomial vector matrix* $M_J^n(x)\rangle$
</polynomialVectorMatrices>

xml for polynomial vector matrix $M_j^n(x)$ $\equiv$
<polynomialVectorMatrix>
<rows>m_j</rows>
<cols>m_j</cols>
<elements>
$\langle$*xml for polynomial vector* $P_{j,11}^n(x)\rangle$
...
$\langle$*xml for polynomial vector* $P_{j,m_j 1}^n(x)\rangle$
...
$\langle$*xml for polynomial vector* $P_{j,1 m_j}^n(x)\rangle$
...
$\langle$*xml for polynomial vector* $P_{j,m_j m_j}^n(x)\rangle$
</elements>

<samplePoints>
<elt>$x_0^{(j)}$</elt>
...
<elt>$x_{d_j}^{(j)}$</elt>
</samplePoints>
<sampleScalings>
<elt>$s_0^{(j)}$</elt>
...
<elt>$s_{d_j}^{(j)}$</elt>
</sampleScalings>
<bilinearBasis>
$\langle$*xml for polynomial* $q_0^{(j)}(x)\rangle$
...
$\langle$*xml for polynomial* $q_{\lfloor d_j/2\rfloor}^{(j)}(x)\rangle$
</bilinearBasis>
</polynomialVectorMatrix>

xml for polynomial vector $P_{j,rs}^n(x)\ \equiv$
<polynomialVector>
$\langle$*xml for polynomial* $P_{j,rs}^0(x)\rangle$
...
$\langle$*xml for polynomial* $P_{j,rs}^N(x)\rangle$
</polynomialVector>

xml for polynomial $a_0 + a_1 x + \ldots a_d x^d\ \equiv$
<polynomial>
<coeff>a_0</coeff>
...
<coeff>a_d</coeff>
</polynomial>

The data can be spread over multiple files. In such cases the different input files should all follow the above format except that $\langle$*xml for objective*$\rangle$ should be omitted from all but one file. The parser will combine the constraints from all the polynomial vector matrices specified in the different files.

Several aspects of the input format are inefficient. Because the matrices are symmetric, `rows` and `cols` are redundant, and most elements are listed twice. Also, XML is extremely verbose.

To improve performance for large inputs, the XML files must first be preprocessed into a more efficient format. Instructions on how to preprocess the input and run SDPB are in `docs/Usage.md`.

3.2 `Mathematica` Interface

A `Mathematica` notebook `SDPB.m`, included in the source distribution, generates files of the form in listing 1 starting from `Mathematica` data. It automatically makes sensible choices for the bilinear bases $q_m^{(j)}(x)$, the sample points $x_k^{(j)}$ and the sample scalings $s_k^{(j)}$.

The `Mathematica` definition of a PMP is slightly different but trivially equivalent to (2.2). It is:

$$\begin{array}{ll} \text{maximize} & a \cdot z \quad \text{over} \quad z \in \mathbb{R}^{N+1}, \\ \text{such that} & \sum_{n=0}^{N} z_n W_j^n(x) \succeq 0 \quad \text{for all } x \geq 0 \text{ and } 1 \leq j \leq J, \\ & n \cdot z = 1. \end{array} \tag{3.1}$$

where $W_j^n(x)$ are matrix polynomials. The normalization condition $n \cdot z = 1$ can be used to solve for one of the components of z in terms of the others. Calling the remaining components $y \in \mathbb{R}^N$, we arrive at (2.2), where $M_j^n(x)$ are linear combinations of $W_j^n(x)$ and b_0, b_n are linear combinations of the a_n. This difference in convention is for convenient use in the conformal bootstrap.

`SDPB.m` defines a function `WriteBootstrapSDP[file, sdp]`, where `file` is the XML file to be written to, and `sdp` has the following form, where the polynomials $Q_{j,rs}^n(x)$ are the elements of $W_j^n(x)$.

Listing 2: Usage of `WriteBootstrapSDP` in `SDPB.m`

```
function call ≡ WriteBootstrapSDP[file, ⟨sdp⟩]

sdp ≡ SDP[⟨objective⟩, ⟨normalization⟩, ⟨positive matrices with prefactors⟩]

objective ≡ {a_0, ..., a_N}

normalization ≡ {n_0, ..., n_N}

positive matrices with prefactors ≡ {
    ⟨positive matrix with prefactor 1⟩,
    ...
    ⟨positive matrix with prefactor J⟩,
  }

positive matrix with prefactor j ≡
  PositiveMatrixWithPrefactor[⟨prefactor⟩,
    {
      {
        {Q^0_{j,11}(x), ..., Q^N_{j,11}(x)}, ..., {Q^0_{j,m_j 1}(x), ..., Q^N_{j,m_j 1}(x)}
      },
      ...
      {
        {Q^0_{j,1m_j}(x), ..., Q^N_{j,1m_j}(x)}, ..., {Q^0_{j,m_j m_j}(x), ..., Q^N_{j,m_j m_j}(x)}
      },
```

```
    }
  ]

prefactor ≡
    DampedRational[c, {p_1,...,p_k}, b, x]
  or
    const
```

The prefactor in PositiveMatrixWithPrefactor is used for constructing bilinear bases and sample scalings. Specifically, if the prefactor is $\chi(x)$, the bilinear basis is a set of orthogonal polynomials with respect to measure $\chi(x)dx$ on the positive real line, and sample scalings are $\chi(x_k)$, where the x_k are sample points. The notebook SDPB.m only deals with damped-rational prefactors because these are relevant to the conformal bootstrap. These stand for

$$\texttt{DampedRational}[c,\ \{p_1,\ldots,p_k\},\ b,\ x] \quad \to \quad c\frac{b^x}{\prod_{i=1}^{k}(x-p_i)}. \tag{3.2}$$

We do not use an exponential-times-rational Mathematica function directly because the DampedRational data structure makes it easier to extract information needed to construct a bilinear basis. The notebook SDPB.m makes a choice of sample points that are reasonable for conformal bootstrap applications.

As an example bootstrap application, the included notebook Bootstrap2dExample.m computes a single-correlator dimension bound for 2d CFTs with a $\mathbb{Z}_2$ symmetry, as in [2].

3.3 An Example

Let's look at an example. Consider the following problem: maximize $-y$ such that

$$1 + x^4 + y\left(\frac{x^4}{12} + x^2\right) \quad \geq \quad 0 \qquad \text{for all } x \geq 0 \tag{3.3}$$

This is an PMP with 1×1 positive-semidefiniteness constraints. We will arbitrarily choose a prefactor of $e^{-x} =$ DampedRational[1,{}, 1/E,x], so that the bilinear basis consists of Laguerre polynomials. The Mathematica code for this example is

Listing 3: Mathematica input for the example (3.3)

```
Module[
  {
    pols = {
      PositiveMatrixWithPrefactor[
        DampedRational[1,{}, 1/E,x],
        {{{1 + x^4, x^4/12 + x^2}}}
      ]
    },
    norm = {1, 0},
```

```
    obj = {0, -1}
  },
  WriteBootstrapSDP["test.xml", SDP[obj, norm, pols]];
];
```

It produces the following XML file

Listing 4: XML file `test.xml` produced by listing 3. Decimals are truncated at 12 digits.

```
<sdp>
  <objective><elt>0</elt><elt>-1</elt></objective>
  <polynomialVectorMatrices>
    <polynomialVectorMatrix>
      <rows>1</rows>
      <cols>1</cols>
      <elements>
        <polynomialVector>
          <polynomial>
            <coeff>1</coeff><coeff>0</coeff><coeff>0</coeff>
            <coeff>0</coeff><coeff>1</coeff>
          </polynomial>
          <polynomial>
            <coeff>0</coeff><coeff>0</coeff><coeff>1</coeff>
            <coeff>0</coeff><coeff>0.0833333333333</coeff>
          </polynomial>
        </polynomialVector>
      </elements>
      <samplePoints>
        <elt>0.017496844815</elt><elt>0.157471603340</elt><elt>0.857345395967</elt>
        <elt>2.117118222694</elt><elt>3.936790083523</elt>
      </samplePoints>
      <sampleScalings>
        <elt>0.982655336118</elt><elt>0.854301072560</elt><elt>0.424286902403</elt>
        <elt>0.120378031823</elt><elt>0.019510742190</elt>
      </sampleScalings>
      <bilinearBasis>
        <polynomial><coeff>1</coeff></polynomial>
        <polynomial><coeff>-1</coeff><coeff>1</coeff></polynomial>
        <polynomial><coeff>1</coeff><coeff>-2</coeff><coeff>0.5</coeff></polynomial>
      </bilinearBasis>
    </polynomialVectorMatrix>
  </polynomialVectorMatrices>
</sdp>
```

4 Internal SDP

To understand the output of `SDPB`, we need a rough understanding of its internal representation of the above PMP as a semidefinite program (SDP). Much more detail is given in [1].

The PMP (2.2) is translated into a dual pair of SDPs of the following form:

$$\begin{array}{ll}\mathcal{D}: & \text{maximize} \quad \mathrm{Tr}(CY)+b_0+b\cdot y \quad \text{over} \quad y\in\mathbb{R}^N,\ Y\in\mathcal{S}^K,\\ & \text{such that} \quad \mathrm{Tr}(A_*Y)+By=c,\ \text{and}\\ & Y\succeq 0.\end{array} \tag{4.1}$$

$$\begin{array}{lll}\mathcal{P}: & \text{minimize} & b_0+c\cdot x \quad \text{over} \quad x\in\mathbb{R}^P,\ X\in\mathcal{S}^K,\\ & \text{such that} & X=\sum_{p=1}^P A_p x_p - C,\\ & & B^T x = b,\\ & & X\succeq 0,\end{array} \tag{4.2}$$

where "$\succeq 0$" means "is positive-semidefinite" and

$$\begin{aligned} c &\in \mathbb{R}^P,\\ B &\in \mathbb{R}^{P\times N},\\ A_1,\ldots,A_P,C &\in \mathcal{S}^K. \end{aligned} \tag{4.3}$$

Here, $\mathcal{S}^K$ is the space of $K\times K$ symmetric real matrices, and $\mathrm{Tr}(A_*Y)$ denotes the vector $(\mathrm{Tr}(A_1Y),\ldots,\mathrm{Tr}(A_PY))\in\mathbb{R}^P$. An optimal solution to (4.1) and (4.2) is characterized by $XY=0$ and also equality of the primal and dual objective functions $\mathrm{Tr}(CY)+b_0+b\cdot y = b_0+c\cdot x$.

The residues

$$\begin{aligned} P &\equiv \sum_i A_i x_i - X - C,\\ p &\equiv b - B^T x,\\ d &\equiv c - \mathrm{Tr}(A_*Y) - By, \end{aligned} \tag{4.4}$$

measure the failure of x, X, y, Y to satisfy their constraints. We say a point $q=(x,X,y,Y)$ is "primal feasible" or "dual feasible" if the residues are sufficiently small,

$$\begin{array}{rrcll}\text{primal feasible:} & \texttt{primalError} & \equiv & \max_{i,j}\{|p_i|,|P_{ij}|\} & < \texttt{primalErrorThreshold};\\ \text{dual feasible:} & \texttt{dualError} & \equiv & \max_i\{|d_i|\} & < \texttt{dualErrorThreshold},\end{array}$$

where `primalErrorThreshold` $\ll 1$ and `dualErrorThreshold` $\ll 1$ are parameters chosen by the user.

An optimal point should be both primal and dual feasible, and have (nearly) equal primal and dual objective values. Specifically, let us define `dualityGap` as the normalized difference between the primal and dual objective functions

$$\begin{aligned} \texttt{dualityGap} &\equiv \frac{|\texttt{primalObjective}-\texttt{dualObjective}|}{\max\{1,|\texttt{primalObjective}+\texttt{dualObjective}|\}},\\ \texttt{primalObjective} &\equiv b_0+c\cdot x,\\ \texttt{dualObjective} &\equiv \mathrm{Tr}(CY)+b_0+b\cdot y. \end{aligned} \tag{4.5}$$

A point is considered "optimal" if

$$\texttt{dualityGap} < \texttt{dualityGapThreshold}, \tag{4.6}$$

where `dualityGapThreshold` $\ll 1$ is chosen by the user.

5 Output of SDPB

5.1 Terminal Output

Listing 5: Output of SDPB for the input file in listing 4

```
$ ./build/sdpb -s test/test --noFinalCheckpoint --dualityGapThreshold=1e-10 --procsPerNode=1
Block Grid Mapping
Node    Num Procs       Cost            Block List
==================================================
0       1               0       {(0,5)}


SDPB started at 2018-Oct-23 15:01:09
SDP directory   : "test/test"
out file        : "test/test.out"
checkpoint in   : "test/test.ck"
checkpoint out  : "test/test.ck"

Parameters:
maxIterations                = 500
maxRuntime                   = 86400
checkpointInterval           = 3600
noFinalCheckpoint            = true
findPrimalFeasible           = false
findDualFeasible             = false
detectPrimalFeasibleJump     = false
detectDualFeasibleJump       = false
precision(actual)            = 400(448)
dualityGapThreshold          = 1e-10
primalErrorThreshold         = 1e-30
dualErrorThreshold           = 1e-30
initialMatrixScalePrimal     = 1e+20
initialMatrixScaleDual       = 1e+20
feasibleCenteringParameter   = 0.1
infeasibleCenteringParameter = 0.3
stepLengthReduction          = 0.9
maxComplementarity           = 1e+100
procsPerNode                 = 1
debug                        = false


          time    mu     P-obj       D-obj      gap         P-err       D-err      P-step   D-step    beta  dim
-----------------------------------------------------------------------------------------------------------------
1            0 1.0e+40  +0.00       +0.00       0.00       +1.00e+20   +2.88e+20   0.811    0.832     0.300  1
2            0 2.7e+39  +1.22e+20   -2.11e+20   1.00       +1.89e+19   +4.84e+19   0.786    0.807     0.300  1
3            0 8.4e+38  +1.27e+20   -3.52e+20   1.00       +4.03e+18   +9.36e+18   0.777    0.794     0.300  1
....
82           0 2.4e-08  +1.84       +1.84       3.22e-08   +4.33e-136  +1.79e-134  1.00     1.00      0.100  1
83           0 2.4e-09  +1.84       +1.84       3.22e-09   +6.99e-136  +4.27e-134  1.00     1.00      0.100  1
84           0 2.4e-10  +1.84       +1.84       3.22e-10   +3.13e-136  +2.20e-134  1.00     1.00      0.100  1
-----found primal-dual optimal solution-----

primalObjective = 1.84026576320318090039117617247403804326445555507439934672628612807738555109679854789376606
1435193498261234053161410573076634281729848342
dualObjective   = 1.84026576308462848033006313255001428991959547020992950642901598201900422505389883109900102
0847250942430618468768821243142382243330096089
dualityGap      = 3.22106791408699658310926876653734497767736974296331635819280399407159685692585961407171090
6339045160904020559956169061628872411051464635e-11
primalError     = 5.00037755305806798164436957351900929576913325025196268071859152540745147053008175998854705
8919012324798467335629270560162565523345346961e-136
dualError       = 1.19696429334187360543910708477679566903833954579370699383899312769928337120183911303802430
2618468190683390894684227272207711446465027672e-133

Saving solution to      : "test/test.out"
```

The output from running `SDPB` on the example problem in section 3.3 is in listing 5. The input, output, and checkpoint files are listed first, followed by various parameters. After each iteration, `SDPB` prints the following:

`time`: The current solver runtime in seconds.

`mu`: The value of the complementarity $\mathrm{Tr}(XY)/K$.

`P-obj`: The primal objective value $b_0 + c \cdot x$.

`D-obj`: The dual objective value $\mathrm{Tr}(CY) + b_0 + b \cdot y$.

`gap`: The value of `dualityGap`.

`P-err`: The primal error $\max_{i,j}\{|p_i|, |P_{ij}|\}$.

`D-err`: The dual error $\max_i\{|d_i|\}$.

`P-step`: The primal step length $\alpha_{\mathcal{P}}$ described in [1].

`D-step`: The dual step length $\alpha_{\mathcal{D}}$ described in [1].

`beta`: The corrector centering parameter β_c described in [1].

`dim`: The dimension of the vector y.

If an optimal solution exists, the primal and dual error will decrease until the problem becomes primal and dual feasible. Then the primal and dual objective functions start to converge, and the complementarity μ decreases until the duality gap becomes smaller than `dualityGapThreshold`.

The terminal output ends with the final values of the primal/dual objectives, primal/dual errors and duality gap.

5.2 Termination

The possible termination reasons for `SDPB` are as follows

`found primal-dual optimal solution`
: Found a solution for x, X, y, Y that is simultaneously primal feasible, dual feasible, and optimal.

`found primal feasible solution`
: Found a solution for x, X that is primal feasible. `SDPB` will only terminate with this result if the option `--findPrimalFeasible` is specified.

`found dual feasible solution`
: Found a solution for y, Y that is dual feasible. `SDPB` will only terminate with this result if the option `--findDualFeasible` is specified.

`primal feasible jump detected`
: A Newton step with primal step length $\alpha_{\mathcal{P}}$ just occurred, without resulting in a primal feasible solution. (Usually this means one should increase `precision`.)

`dual feasible jump detected`
: A Newton step with dual step length $\alpha_{\mathcal{D}}$ just occurred, without resulting in a dual feasible solution. (Usually this means one should increase `precision`.)

`maxIterations exceeded`
: `SDPB` has run for more iterations than specified by the option `--maxIterations`.

`maxRuntime exceeded`
: `SDPB` has run for longer than specified by the option `--maxRuntime`.

`maxComplementarity exceeded`
: $\mu = \mathrm{Tr}(XY)/\dim(X)$ exceeded the value specified by `--maxComplementarity`. This might indicate that the problem is unbounded and no optimal solution will be found.

When using `SDPB` to determine primal or dual feasibility, one can specify the options `--findPrimalFeasible` or `--findDualFeasible`. This will cause the solver to terminate immediately once the primal or dual errors are sufficiently small. This often occurs immediately after the primal or dual step lengths become equal to 1. A step length of 1 means that the solver has found a Newton step that exactly solves the primal or dual constraints, while preserving positive-semidefiniteness of X, Y. Sometimes a step length of 1 does not result in sufficiently small primal/dual errors. This is indicative of numerical instabilities and usually means `precision` should be increased. The options `--detectPrimalFeasibleJump` and `--detectPrimalFeasibleJump` cause `SDPB` to terminate if a step length of 1 occurs without resulting in primal/dual feasibility. If desired, one can then restart the solver with a higher value of `precision`.

5.3 Output File

Listing 6: Contents of the output file `test.out` corresponding to listing 4. Decimal expansions have been truncated for brevity. `Mathematica` uses `*^` instead of the character `e` for scientific notation. Thus, the output format is not quite suitable for import into `Mathematica` without modification. This could be changed in future versions.

```
terminateReason = "found primal-dual optimal solution";
primalObjective = 1.840265763203;
dualObjective   = 1.840265763084;
dualityGap      = 3.221067914086e-11;
primalError     = 4.263251669979e-136;
dualError       = 1.421540011331e-133;
runtime         = 0.1612241119146347045898437 5;
y = {-1.840265763084};
x = {0.4523538794795, -0.803480855768, 2.460542537885, 0.361240154722, -0.094037214700};
```

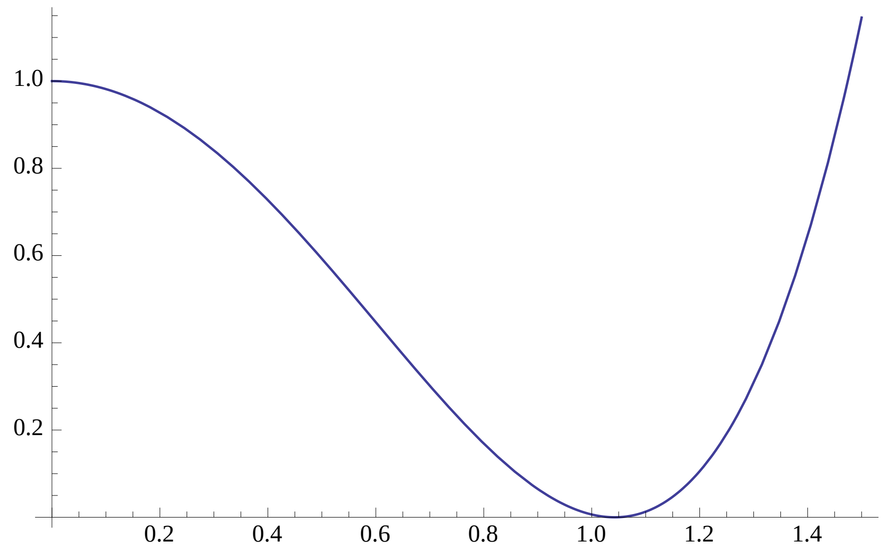

Figure 1: A plot of $1 + x^4 + y\left(\frac{x^4}{12} + x^2\right)$ with $y = -1.840265763084$ equal to its optimal value. The zero near $x = 1$ shows that $-y$ cannot be further increased without violating the positivity constraint.

The output file `test.out` corresponding to listing 4 is shown in listing 6. It includes the reason for termination, the final primal/dual objective values, the final duality gap, the final primal/dual errors, the total runtime, and the vectors y and x.[1]

The value of y gives the solution to our optimization problem. The function

$$1 + x^4 + (-1.840265763084)\left(\frac{x^4}{12} + x^2\right) \tag{5.1}$$

is plotted in figure 1. The zero near $x = 1$ shows that y is optimal.

5.4 Checkpoints

Every `checkpointInterval`, `SDPB` saves a new checkpoint in a directory with the `.ck` extension. `SDPB` also saves a checkpoint after termination, provided the option `--noFinalCheckpoint` is not specified.

A checkpoint file encodes the values of x, X, y, Y. If `SDPB` detects an existing checkpoint file on startup, it will use those values of x, X, y, Y as initial conditions in the solver. Thus, `SDPB` can be stopped and started at will without losing progress.

A typical workflow for long-running computations on shared machines is to specify a moderate `checkpointInterval` (e.g. one hour) and a somewhat larger `maxRuntime` (e.g. 12 hours). `SDPB` will terminate after 12 hours and can then be restarted without losing

[1] To include the matrices X, Y as well, uncomment the lines "`// ofs << "Y = " << Y << ";\n";`" and "`// ofs << "X = " << X << ";\n";`" in the source file `SDPSolverIO.cpp`, and recompile `SDPB`.

progress. If SDPB is killed prematurely, then at most 1 hour of progress will be lost. This pattern of restarting gives other users chances to run their processes. It can be sustained indefinitely, allowing extremely long computations.

6 Attribution

If you use SDPB in work that results in publication, please cite [1]. Depending on how SDPB is used, the following sources might also be relevant:

- The first use of semidefinite programming in the bootstrap [3].
- The generalization of semidefinite programming methods to arbitrary spacetime dimension [4].
- The generalization of semidefinite programming methods to arbitrary systems of correlation functions [5].

7 Acknowledgements

SDPB makes extensive use of the parallel linear algebra library Elemental [11], the Boost C++ libraries [9], the libxml2 library [10], and the multiprecision libraries GMP [12], MPC [14], and MPFR [13].

SDPB was partially based on the solvers SDPA and SDPA-GMP [6–8], which were essential sources of inspiration and examples.

Thanks to Filip Kos, David Poland, and Alessandro Vichi for collaboration in developing semidefinite programming methods for the conformal bootstrap and assistance testing SDPB. Thanks to Amir Ali Ahmadi, Hande Benson, Pablo Parrilo, and Robert Vanderbei for advice and discussions about semidefinite programming.

I am supported by DOE grant number DE-SC0009988 and a William D. Loughlin Membership at the Institute for Advanced Study.

References

[1] David Simmons-Duffin, "A Semidefinite Program Solver for the Conformal Bootstrap," arXiv:1502.02033 [hep-th].

[2] V. S. Rychkov and A. Vichi, "Universal Constraints on Conformal Operator Dimensions," Phys. Rev. D **80**, 045006 (2009) arXiv:0905.2211 [hep-th].

[3] D. Poland, D. Simmons-Duffin and A. Vichi, "Carving Out the Space of 4D CFTs," JHEP **1205**, 110 (2012) arXiv:1109.5176 [hep-th].

[4] F. Kos, D. Poland and D. Simmons-Duffin, "Bootstrapping the $O(N)$ vector models," JHEP **1406**, 091 (2014) arXiv:1307.6856 [hep-th].

[5] F. Kos, D. Poland and D. Simmons-Duffin, "Bootstrapping Mixed Correlators in the 3D Ising Model," JHEP **1411**, 109 (2014) arXiv:1406.4858 [hep-th].

[6] M. Yamashita, K. Fujisawa, M. Fukuda, K. Nakata, and M. Nakata, "A high-performance software package for semidefinite programs: SDPA 7," Research Report B-463, Dept. of Mathematical and Computing Science, Tokyo Institute of Technology, Tokyo, Japan (2010).

[7] M. Yamashita, K. Fujisawa, and M. Kojima, "Implementation and evaluation of SDPA 6.0 (SemiDefinite Programming Algorithm 6.0)," Optimization Methods and Software" 18 491-505 (2003).

[8] M. Nakata, "A numerical evaluation of highly accurate multiple-precision arithmetic version of semidefinite programming solver: SDPA-GMP, -QD and -DD.," 2010 IEEE International Symposium on Computer-Aided Control System Design (CACSD), 29-34 Sept 2010.

[9] C++ Standards Committee Library Working Group and other contributors, "BOOST C++ Libraries," http://www.boost.org.

[10] Gnome Project, Libxml2, http://www.xmlsoft.org/

[11] J. Poulson, B. Marker, R. van de Geijn, J. Hammond, and N. Romero, "Elemental: A new framework for distributed memory dense matrix computations, ACM Transactions on Mathematical Software," ACM Trans. Math. Softw. 39 2 13:1-24 (2013), doi:10.1145/2427023.2427030

[12] The GNU Multiprecision Library, https://gmplib.org/

[13] The GNU MPFR Library, https://www.mpfr.org/

[14] The GNU MPC Library, http://www.multiprecision.org/mpc