BKsolver: numerical solution of the Balitsky–Kovchegov nonlinear integro-differential equation

Rikard Enberg

Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, USA, and Centre de Physique Théorique, École Polytechnique, 91128 Palaiseau, France

Abstract

BKsolver is a program that solves the Balitsky–Kovchegov equation, which is a non-linear integro-differential equation that describes the behaviour of QCD scattering amplitudes in the high–energy limit using a mean field approximation of the saturation physics. This manual outlines the algorithm and describes the use of the program. The numerical method is based on Runge–Kutta integration of the system of differential equations obtained from Chebyshev approximation of the integral, which in turn uses a Fast Fourier Transform to sum the Chebyshev coefficients. The program can easily be modified to solve other one-dimensional non-linear integro-differential equations.

1 The Balitsky–Kovchegov equation

We want to solve the BK equation [1] in its momentum space form

$$\frac{\partial \phi(k)}{\partial Y} = \bar{\alpha}_s \int_0^\infty \frac{\mathrm{d}\ell^2}{\ell^2} \left[\frac{\ell^2 \phi(\ell) - k^2 \phi(k)}{|k^2 - \ell^2|} + \frac{k^2 \phi(k)}{\sqrt{4\ell^4 + k^4}} \right] - \bar{\alpha}_s \phi^2(k) \tag{1}$$

where it is understood that $\phi(k) = \phi(k, Y)$. In discussing travelling waves (see [3] and references therein) it is natural to change to the variable $L = \ln k^2, L' = \ln \ell^2$, which casts the integral on the right hand side above on the form

$$\bar{\alpha}_s \int_{-\infty}^{+\infty} \mathrm{d}L' \left(\frac{\tilde{\phi}(L') - e^{L-L'} \tilde{\phi}(L)}{|1 - e^{L-L'}|} + \frac{e^{L-L'} \tilde{\phi}(L)}{\sqrt{4 + e^{2(L-L')}}} \right) . \tag{2}$$

For the numerical solution the range of L must be made finite [2]:

$$L = (M_1 + M_2)u - M_1 \tag{3}$$

$$L' = (M_1 + M_2)v - M_1 \tag{4}$$

where $u, v \in [0,1]$, and $-M_1$ and M_2 are the lower and upper bounds on $\log k^2 = (M_1 + M_2)u - M_1$.[1]

Thus we get

$$\frac{\partial \hat{\phi}(u)}{\partial Y} = (M_1 + M_2) \int_0^1 \mathrm{d}v \left[\frac{\hat{\phi}(v) - \mathrm{e}^{(M_1+M_2)(u-v)}\hat{\phi}(u)}{|\mathrm{e}^{(M_1+M_2)(u-v)} - 1|} \right.$$
$$\left. + \frac{\hat{\phi}(u)}{\sqrt{1 + 4\mathrm{e}^{-2(M_1+M_2)(u-v)}}} \right] - \bar{\alpha}_s \hat{\phi}^2(u). \tag{5}$$

The integrand seems to be singular along the line $u = v$, but this is not a real singularity, and written as above the first term vanishes for $u = v$.

BKsolver also includes another integral kernel that can be optionally studied. This is the two-pole approximation, where the BFKL kernel in Mellin space, $\chi(\gamma) = \psi(1) - \psi(\gamma) - \psi(1-\gamma)$, is simply approximated by keeping only the two poles at $\gamma = 0$ and $\gamma = 1$ plus a constant term adjusted to give the same pomeron intercept:

$$\chi(\gamma) \simeq \frac{1}{\gamma} + \frac{1}{1-\gamma} + 4(\ln 2 - 1)\ . \tag{6}$$

This leads to an integral kernel on the form

$$\int_L^{+\infty} \mathrm{d}L'\, \tilde{\phi}(L') + \int_{-\infty}^{L} \mathrm{d}L'\, e^{L'-L}\, \tilde{\phi}(L') + 4(\ln 2 - 1)\tilde{\phi}(L)\ . \tag{7}$$

Algorithm for the numerical solution

The method is based on properties of the Chebyshev polynomials. The Chebyshev polynomial $T_n(x)$ has zeros in the points

$$x_k = \cos\left(\frac{\pi(k - \frac{1}{2})}{n}\right), \qquad k = 1, \ldots, n \tag{8}$$

and extrema with values ± 1 in the points

$$\widetilde{x}_k = \cos\left(\frac{\pi k}{n}\right), \qquad k = 0, \ldots, n. \tag{9}$$

The polynomials in these points obey certain orthogonality relations that allow the normal Chebyshev expansion of a function $f(x)$. In fact to compute the coefficients in the expansion, values of $f(x)$ are only needed at either at the points x_k or at $\widetilde{x}_k$ depending on the choice of definition of the series.

Therefore a natural choice of grid points for the u-grid is precisely the points $\widetilde{x}_k$ above, although here we will only need them in the interval $[0,1]$. In [4, 5] it is pointed out that

[1]For most purposes in [3] $M_1 = 20$ and $M_2 = 138$ were used, but going to $Y = 250$ much larger grids were needed.

one can get the integral of the function $f(x)$ without actually computing the Chebyshev coefficients, but by substituting the definition into the Chebyshev series and integrating. Thus for Chebyshev approximation of order N we have

$$I \equiv \int_a^b f(t)\,\mathrm{d}t \approx \sum_{k=0}^{N}{}'' f(\widetilde{x}_k)\,\frac{2}{N}\sum_{j=0}^{N}{}'' T_j(\widetilde{x}_k)\int_a^b T_j(t)\,\mathrm{d}t \tag{10}$$

where the double prime on the sum denotes that the first and last terms are divided by 2.

The integral of T_j is for $(a,b)=(0,1)$ given by

$$\int_0^1 T_j(t)\,\mathrm{d}t = \begin{cases} \dfrac{j\sin(j\pi/2)-1}{j^2-1} & \text{for } j \neq 1 \\ 1/2 & \text{for } j = 1. \end{cases} \tag{11}$$

Thus the method consists of the following:

1. Discretize u and v at the points $\widetilde{x}_k$. Start with a suitable initial condition at $Y=0$.
2. The function ϕ is only needed at the u-grid points in each rapidity step. Perform the v-integral by using the tabulated ϕ-values from the previous rapidity step in formula (10).
3. This gives a large system of first order ordinary differential equations (ODE's). Solve this numerically for each step in rapidity.

The advantage of this procedure is that the grid can be relatively small; only 256 points in momentum were used for most results presented in [3]. Therefore very large systems of ODE's are avoided.

The program uses the Gnu Scientific Library [6] for ODE solving, interpolation and some other useful tasks. The system of ODE's is solved using an embedded Runge–Kutta–Fehlberg 4,5 method. For large N the sum (10) is very time consuming to perform, and is prone to numerical errors. Numerical Recipes [7] mentions that this type of sum is efficiently done using a cosine Fast Fourier Transform (a Discrete Cosine Transform (DCT)). I have therefore incorporated a DCT routine by Ooura [8], which decreases the running time substantially. Moreover, for large rapidities the naive summation gives incorrect results due to inaccuracies.

The resulting program is fast—the solution for a grid of 256 points in k, $N = 2\cdot 256$, and rapidity up to $Y = 50$ takes about 20–25 seconds on a 2 GHz Pentium Mobile laptop running GNU/Linux.

2 Description of the program

The source code and this manual, as well as any additional documentation and results can be found at `http://www.isv.uu.se/~enberg/BK/`. If you use this program please cite [3] and this manual.

The main program, BKsolver, solves the BK equation starting from a specified initial condition at rapidity $Y = 0$, for a specified range of momentum k and rapidity Y, and writes the result to a file (or one file for each step in rapidity if desired) as a two-dimensional grid in k and Y. All settings and input is read from a configuration file provided by the user, unless one uses the default values specified in the code. There is also a separate program that reads the output file from BKsolver and uses the results to compute the saturation scale $Q_s(Y)$ and the logarithmic derivative $\partial \log Q_s^2(Y)/\partial Y$ as functions of Y, as well as the reduced front shape $\psi(k, Y)$ and the functions $\mathcal{F}(Y)$ and $\mathcal{G}(Y)$ as defined in [3].

The program uses the GNU Scientific Library (GSL) [6] to perform the numerical solution of the ODE's, for spline interpolation, and for some other minor utilities. It also uses the parsecfg package [9] to interpret the configuration file, and the FFT package [8] to perform the necessary Discrete Cosine Transforms. The latter two packages are supplied with the BKsolver distribution, while GSL must be installed separately (and I highly recommend using GSL for your other computing tasks!).

All code is written in ISO-compliant C and compiles cleanly with the GNU C compiler gcc when invoking most of the warning flags. It has also been thoroughly tested on the GNU/Linux operating system with programs such as Splint and Valgrind.

Using the program

BKsolver consists of a main program that reads a configuration file, if provided, and sets up the problem to be solved. It prints the parameters used to the terminal and runs the numerical solution. The results of the solution are written to the specified file(s).

To perform a solution the provided source code should be compiled and linked with the GNU Scientific Library. The problem to be solved should be defined in the configuration file that is read by the program. There is an example configuration file provided with the package, which is a good starting point for changes. Any parameters that are not given values in the configuration file will retain their default values—see the Appendix for a list of parameters and default values.

The program is started with a configuration file by typing (assuming the code is compiled to an executable called `bk`):

```
./bk config-filename
```

and without using a configuration file by simply omitting the file name. Any additional arguments on the command line will be ignored. Parameter values are given by specifying

```
PARAMETER_NAME = parameter-value
```

where the parameter value is a number, a string in quotations marks or, for boolean values, "`yes`", "`no`", "`true`", "`false`", etc. The parameter names are not case sensitive. Lines starting with # are comments.

The parameters that can be changed are listed in the Appendix. There are parameters to change the grids, which integral kernel is used, the initial condition, filenames, accuracy etc. The initial condition can either be one of the simple predefined forms in the program, or it can be read and interpolated from a table in a file as a function of k. A file with the McLerran–Venugopalan (MV) initial condition, used in [3], is provided under the name `mv-k.front.dat`. To use your own favourite function, just write a program to tabulate it in the desired k-range.

The result of the solution will be written to disk, either one file containing a two-dimensional grid in a form suitable for plotting with `gnuplot`, or to one file for each rapidity, with a one-dimensional table in each file.

3 Modifying and distributing the program

BKsolver is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This holds with the exception of the Fast Fourier Transform package which is distributable but not under GPL. The GPL is included in the download.

There are many things that can be modified, the most obvious being that it is possible to solve other integro-differential equations. The integral kernel is defined in the function `double integrand(int vi, int ui)` and the non-linear term is added in the function `double cheb_approx(int ui)`. The latter function is where any constant terms should be added, such as for the two-pole kernel. It is easy to add your own equation to these routines.

Finally, BKsolver is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

A Parameters

Parameter	Default	Description
`KMAX`	256	number of grid points in k
`Y_GRID_MAX`	10	number of points in Y
`DELTA_Y`	5.0	rapidity step
`BK_KERNEL`	0	kernel used for evolution =0: normal BK kernel =1: two-pole approximation of kernel
`IC_CHOICE`	0	initial condition at $Y = 0$ =0: read from file (MV initial condition provided) =1: step function =2: "smoothened step function" =3: Gaussian =4: $\sim 1/\sqrt{k}$
`IC_FILE`	`mv-k.front.dat`	filename if initial condition is read from file
`IC_POSITION`	1.0	position in k of the initial condition
`ALPHA_RUNNING`	`no`	choose fixed or running strong coupling
`ALPHA_BAR`	0.2	value of $\bar{\alpha}_s$ for fixed coupling
`N_FLAVOUR`	3	n_f for running coupling
`LAMBDA_QCD`	0.2	Λ_{QCD} for running coupling
`ALPHA_FREEZE`	0.5	value at which the running coupling is frozen
`PRINT_SPLIT`	`no`	print results in separate files for each rapidity?
`LN_K_MIN`	20.0	M_1 (see Eqs. (3,4))
`LN_K_MAX`	138.0	M_2 (see Eqs. (3,4))
`ODE_ACCURACY`	10^{-3}	accuracy of ODE solver (see GSL manual [6])
`ODE_STEPSIZE`	10^{-3}	initial step size of ODE solver (see [6])
`OUTPUT_FILE`	`bkresult`	filename of the output file (`.dat` will be added). For `PRINT_SPLIT=yes` this is the base name for file name generation.

Acknowledgements

This program was developed for the research presented in [3] together with Krzyztof Golec-Biernat and Stéphane Munier and I thank them for a fruitful collaboration and many discussions. This work was to a large part carried out when I was at the Centre de Physique Théorique, École Polytechnique (Unité mixte de recherche du CNRS, no. 7644).

References

[1] I. Balitsky, Nucl. Phys.**B463** (1996) 99; Phys. Rev. Lett. **81** (1998) 2024; Phys. Lett. **B518** (2001) 235; Y.V. Kovchegov, Phys. Rev. **D60** (1999) 034008; Phys. Rev. **D61** (2000) 074018.

[2] M. Braun, Eur. Phys. J. C **16** (2000) 337

[3] R. Enberg, K. Golec-Biernat and S. Munier, arXiv:hep-ph/0505101.

[4] S. E. El-gendi, Comput. J. **12** (1969) 282

[5] B. Mihaila and I. Mihaila, J. Phys. A: Math. Gen. **35** (2002) 731

[6] M. Galassi et al., "GNU Scientific Library Reference Manual" 2nd Ed. (Network Theory, 2003), http://www.gnu.org/software/gsl/

[7] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, "Numerical Recipes in C" 2nd Ed., (Cambridge University Press, 1992).

[8] T. Ooura, "General Purpose FFT (Fast Fourier/Cosine/Sine Transform) Package", http://momonga.t.u-tokyo.ac.jp/˜ooura/fft.html (2001).

[9] Y. Ninomiya, "parsecfg – a library for parsing a configuration file", http://www.icewalkers.com/Linux/Software/56270/parsecfg.html (2001).