UCRL-MA-147919

HBTprogs Version 1.0

D. A. Brown, P. Danielewicz

March 15, 2002

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

DISCLAIMER

This work was performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

HBTprogs
Version 1.0

D.A. Brown[1], P. Danielewicz[2]

March 15, 2002

[1]email:brown170@llnl.gov, L-414, Livermore National National Laboratory, P. O. Box 808, Livermore, CA 94550
[2]email:danielewicz@nscl.msu.edu, Michigan State University, East Lansing, Michigan 48824

Abstract

This is the manual for a collection of programs that can be used to invert angled-averaged (i.e. one dimensional) two-particle correlation functions. This package consists of several programs that generate kernel matrices (basically the relative wavefunction of the pair, squared), programs that generate test correlation functions from test sources of various types and the program that actually inverts the data using the kernel matrix.

Contents

Chapter 1

INTRODUCTION

This manual outlines the set of programs that we developed to invert angle-averaged two-particle correlation functions. In the following sections, we will outline the purpose of the code, how it works, how to install and run the code, and indicate how to extend it.

You are free to use the codes in your analysis provided that you reference either this manual or references [1, 2, 3] in any publications using results obtained with these codes. We are not responsible for any misuse of these codes. Feel free to contact either of us if you have any questions, comments or suggestions. Our emails are David Brown at **brown170@llnl.gov** and Pawel Danielewicz at **danielewicz@nscl.msu.edu**. If you have bug-fixes or add new features or kernels to your version of the code please contact David Brown, so that we can incorporate the changes in future releases.

The outline of this manual is as follows. First, for the remainder of this chapter we will explain the theory behind the imaging codes. Second, in Chapters 2–4, we will explain what is needed to run the codes. This includes explaining how to install and run the codes (Chap. 2), an example calculation (Chap. 3) and a description of the input options (Chap. 4). Next, in Chapters 5–7 we explain more about how to run the various codes and control scripts. In Chap. 5 we elaborate on the imaging codes, in Chap. 6 we explain how the kernel generators work, and in Chap. 7 we detail two ways to test the imaging codes. Following this, Chap. 8 is a list of Frequently Asked Questions. Finally, we will conclude in Chap. 9. There are also three appendices listing various files in the distribution.

1.1 Basic Theory

1.1.1 The Koonin-Pratt Equation

Our starting point is the ratio of the two particle spectrum to two uncorrelated single particle spectra, a.k.a. the two particle correlation function, often measured in heavy-ion reactions:

$$C_{\mathbf{P}}(\mathbf{q}') = \frac{E_1 E_2 d^6N / d^3p_1\, d^3p_2}{E_1(d^3N/d^3p_1)(E_2 d^3N/d^3p_2)} \qquad (1.1)$$

Here $\mathbf{P} = \mathbf{p_1} + \mathbf{p_2}$ is the total momentum of the pair and $\mathbf{q}' = \frac{1}{2}(\mathbf{p_1'} - \mathbf{p_2'})$ is the relative momentum of the pair in the pair center of mass (CM) frame (denoted by the primes).

Extracting the source function for a pair of identical particles, $S_{\mathbf{P}}(\mathbf{r}')$, begins by noting that $S_{\mathbf{P}}(\mathbf{r}')$ is related to this experimentally measured two-particle correlation, $C_{\mathbf{P}}(\mathbf{q}')$ (or $\mathcal{R}_{\mathbf{P}}(\mathbf{q}')$, its deviation from 1), through the so-called Koonin-Pratt equation [4, 5]:

$$\mathcal{R}_{\mathbf{P}}(\mathbf{q}') \equiv C_{\mathbf{P}}(\mathbf{q}') - 1 = \int d\mathbf{r}'\, K(\mathbf{q}', \mathbf{r}')\, S_{\mathbf{P}}(\mathbf{r}')\,. \qquad (1.2)$$

Thus, "imaging the source" means somehow inverting this equation. In (1.2), and the kernel of the integral equation is

$$K(\mathbf{q}', \mathbf{r}') = |\Phi^{(-)}_{\mathbf{q}'}(\mathbf{r}')|^2 - 1. \qquad (1.3)$$

The wavefunction, $\Phi^{(-)}$, describes the propagation of the pair from a relative separation of $\mathbf{r}'$ in the pair CM to the detector with relative momentum $\mathbf{q}'$. The source function itself is the probability of emitting the pair a distance of $\mathbf{r}'$ apart, in the pair CM frame.

1.1.2 The Source Function

In order to get a better understanding of what the source function is, consider the emission function $D(r, \mathbf{p})$. The emission function is the probability of emitting a single particle at space-time point r with momentum $\mathbf{p}$. We may write it in terms of the particle emission rates:

$$D(r, \mathbf{p}) = \frac{Ed^7N}{d^4r\, d^3p} \Big/ \frac{Ed^3N}{d^3p}. \tag{1.4}$$

This is actually not a true probability as it is a Wigner function, however we may identify the emission rates with the freeze-out distributions in a transport code. In fact, this is what is done in Scott Pratt's CRAB code [6].

Next, we may define the relative distance distribution, $d(r, \mathbf{P}/2)$, in terms of the emission function:

$$d(r, \mathbf{P}/2) = \int d^4R\, D(R + r/2, \mathbf{P}/2) D(R - r/2, \mathbf{P}/2) \tag{1.5}$$

The relative distance distribution is the probability of emitting a pair with a space-time separation r. Each particle has approximately momentum $\mathbf{P}/2$ (in the smoothness approximation, the slight difference between using $\mathbf{p_1}$, $\mathbf{p_1}$ and $\mathbf{P}/2$ is ignored). In order to write down Eq. (1.5), we need to assume the each particle comprising the pair is independently emitted. This "chaoticity" assumption is not needed for the imaging to work, but is needed in order to make contact with semi-classical transport models.

Finally, we may define the source function. To do this, we boost to the pair CM frame (noting that $\mathbf{P}'/2 = 0$ in this frame) and integrate out the CM time:

$$S_{\mathbf{P}}(\mathbf{r}') = \int dr'_0\, d_{\mathbf{P}}(r', 0). \tag{1.6}$$

This time integral is a reflection of the fact that the squared wavefunction of the pair is independent of the relative emission time in the pair CM. This time integral does have dramatic consequences, for example we can not distinguish between the two scenarios pictured in Fig. 1.1. As a final comment, the source function is normalized:

$$\int d^3r\, S_{\mathbf{P}}(\mathbf{r}) = 1. \tag{1.7}$$

This is simply a statement of conservation of probability.

1.1.3 Recasting the Problem in 1d

A one dimensional correlation function is usually tabulated as a function of $q_{inv} = \sqrt{-(p_1 - p_2)^2}/2$. Fortunately, the 0^{th} component of the relative momentum vanishes in the pair CM frame for identical particles giving $q_{inv} = |\mathbf{q}'| = q$. Thus, we may angle average the 3d Koonin-Pratt equation to obtain the same equation for 1d correlations. We will now illustrate this.

First, expand the full 3d source and correlation in spherical harmonics, giving

$$S(\mathbf{r}) = \sqrt{4\pi} \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} S_{\ell m}(r) Y_{\ell m}(\hat{r}) \tag{1.8}$$

and

$$C(\mathbf{q}) = \sqrt{4\pi} \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} C_{\ell m}(q) Y_{\ell m}(\hat{q}) \tag{1.9}$$

From here onward, we will suppress the $\mathbf{P}$ subscript in all quantities. The 0^{th} term in both expansions is the angle average of the respective function. Note that the kernel only depends on $|\mathbf{q}|$, $|\mathbf{r}|$, and γ (the angle between $\mathbf{q}$ and $\mathbf{r}$) [1]. This allows us to expand the kernel in Legendre polynomials:

$$K(\mathbf{q}, \mathbf{r}) = \sum_{\lambda=0}^{\infty} (2\lambda + 1) K_\lambda(q, r) P^\lambda(\cos\gamma) \tag{1.10}$$

With this, the integrals can easily be done with the aid of the spherical harmonic addition theorem. In the end, we find

$$\mathcal{R}(q) = C(q) - 1 = 4\pi \int_0^\infty dr r^2 K_0(q, r) S(r). \tag{1.11}$$

and this is the equation we will image. Here, the angle-averaged kernel is

$$K_0(q, r) = \frac{1}{2} \int d(\cos\gamma) \left(\left| \Phi_{\mathbf{q}}^{(-)}(\mathbf{r}) \right|^2 - 1 \right). \tag{1.12}$$

The explicit expressions of the angle-averaged kernels for each pair type are shown in Chap. 6.

1.1.4 Limitations of the Formalism

There are several limitations of the Koonin-Pratt formalism that may affect either the imaging results or the interpretation of the resulting images. Unfortunately, these issues have not been fully mapped out, so at best we can refer you to the literature if you need more information:

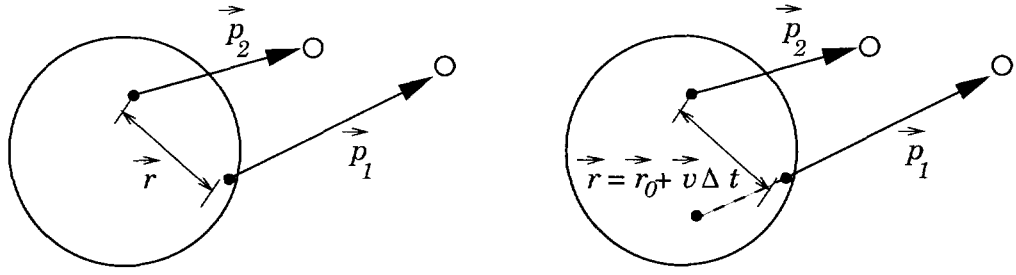

Figure 1.1: Two emission scenarios that give the same separation of emission points. On the left, the pair is emitted simultaneously with a relative separation $\mathbf{r}$. On the right, the first particle is emitted with a relative velocity $\mathbf{v}$ and the second particle is emitted Δt later a distance of $\mathbf{r_0}$ from the first particle. The combined spatio-temporal separation of emission is $\mathbf{r} = \mathbf{r_0} + \mathbf{v}\Delta t$, giving a separation identical to that in the scenario on the left.

1. Third body Coulomb effects [7, 8].
2. Position-momentum correlations (e.g. flow, opacity, etc.) [9].
3. Three(and higher) body quantum statistical effects [10, 11, 12, 13].
4. Smoothness approximations [14, 15, 16].
5. "Chaoticity," in other words, the independent emission hypothesis [17].

Also, there are several review articles that may prove helpful [18, 19, 20].

1.2 Inverting the Koonin-Pratt Equation

In this section, we describe the mathematical considerations behind our inversion code. Although we will transform the inversion of the Koonin-Pratt equation to a simple linear matrix inversion problem, there are several factors that will complicate our work. First, our the data contains both noise and uncertainty. Both must be dealt with using a least-square approach to the inversion. Since the kernel is the pair wavefunction squared, it is basically a distorted sine function. The distortion arises from the pair interaction potential. The similarity of our integral transform to a Fourier Transform problem means that we must pay attention to sampling issues. In particular, we must be careful in choosing our representation of the source. We will use Basis splines (a.k.a. b-splines) to represent our source because we can adjust the knots of the b-splines in order to optimize the resolution. Finally, since this inversion problem is ill-posed, we will use constraints to help stabilize the inversion. All of this is discussed in Ref. [3] in more detail.

1.2.1 Recasting the Equation

In our calculations, we expand the imaged source in a function basis:

$$S(r) = \sum_{i=1}^{N_M} S_i B_i(r). \tag{1.13}$$

Here S_i are the N_M source coefficients of the source with the basis functions $B_i(r)$. In this basis, the error on $S(r)$ is given by

$$\Delta S(r) = \sqrt{\sum_{ij} \Delta^2 S_{ij} B_i(r) B_j(r)} \tag{1.14}$$

where $\Delta^2 S_{ij}$ is the covariance matrix of source coefficients determined during the inversion process.

The experimental data is also binned (with the number of bins being N_D) and therefore we must also average the kernel over these bins. Our inversion problem then reduces to the following equation:

$$\mathcal{R}_i \equiv \mathcal{R}(q_i) = \sum_{j=1}^{N_M} K_{ij} S_j, \tag{1.15}$$

or in matrix notation:

$$\mathcal{R} = K \cdot \mathbf{S}. \tag{1.16}$$

Here, the kernel matrix is

$$K_{ij} = \frac{4\pi}{\Delta q} \int_{q_i - \Delta q_i/2}^{q_i + \Delta q_i/2} dq \int_0^\infty dr\, r^2 K(q,r) B_j(r). \tag{1.17}$$

We call the center of the i^{th} bin q_i and the width of the i^{th} bin is Δq_i. Our source vector is made of the S_j coefficients of the basis function representation of the source and our data vector is made of the $\mathcal{R}_i$ correlation values.

1.2.2 A Least-squares Solution

Once we have chosen a representation of the source and converted the inversion problem into the matrix inversion of Eq. (1.16), we proceed as in Refs. [1, 2] and extract the source. To obtain the coefficients of the source, we seek the source that minimizes the χ^2:

$$\chi^2 = (K \cdot \mathbf{S} - \mathcal{R}^{\mathbf{obs}})^T \cdot (\Delta^2 \mathcal{R})^{-1} \cdot (K \cdot \mathbf{S} - \mathcal{R}^{\mathbf{obs}}) \tag{1.18}$$

Here $\mathcal{R}^{\mathbf{obs}}$ is the vector of data values and $\Delta^2\mathcal{R}$ is the full data covariance matrix. Typically $\Delta^2\mathcal{R}$ is assumed to be diagonal, although it need not be.

The source that minimizes the χ^2 is:

$$\mathbf{S} = \Delta^2 S \cdot K^T \cdot (\Delta^2 \mathcal{R})^{-1} \cdot \mathcal{R}^{\mathbf{obs}} \tag{1.19}$$

The covariance matrix of this source is:

$$\Delta^2 S = (K^T \cdot (\Delta^2 \mathcal{R})^{-1} \cdot K)^{-1} \tag{1.20}$$

When we image, we are really finding a probability density for the source given the correlation data rather than the source itself. The set of source coefficients and the covariance matrix of the source characterize the height and width of this probability distribution. In the end, we use the source coefficients as an estimator of the true source.

1.2.3 Fourier Theory Considerations

Although we have a solution to the imaging problem in Eqs. (1.19) and (1.20), the solution is incomplete with out specifying the basis of the source. Given that the identical particle kernels in Eq. (1.2) or (1.11) are Fourier transform kernels at large distances, we expect our transforms to behave like Fourier transforms. Thus, if we expand the source in box-splines, i.e. discretize it, then Eq. (1.16) is nearly a finite Fourier transform. In this case, the Sampling Theorem says that the binning in q and r spaces are related:

$$\Delta r = \frac{\hbar c \pi}{q_{max}} \text{ and } \Delta q = \frac{\hbar c \pi}{r_{max}}. \tag{1.21}$$

Here $q_{max} = N\Delta q$, $r_{max} = N\Delta r$ and $N_M = N_D \equiv N$ is the number of bins in both the r and q spaces.

Using these relations, we may get a feeling for how structures in the data affect the imaged source. For example, the low-q structure in the data sets the large length scale behavior of the source. Conversely, the high-q portion of the data sets the short length scale behavior of the source and therefore sets the size of the smallest features features we could hope to resolve in the source. For example, if the correlation dies off around a $q \approx 80$ MeV/c, then we should not expect to resolve structure smaller than $\Delta r \approx 8$ fm. Owing to the fact that our kernel is not a trigonometric function in general, these estimates are qualitative at best.

1.2.4 Representing the Source

Rather than use the simple box-spline basis, we choose to represent the source function in a Basis spline (a.k.a. b-spline) basis. Several b-splines are pictured in Fig. 1.2. B-splines are piecewise polynomials and are continuous up to the degree of these polynomials. This basis has many features that make it ideal for the imaging problem:

- It can efficiently encode complicated functions.
- Basis functions can be evaluated quickly.
- The degree of continuity of the b-splines can be adjusted in several ways.
- Natural generalization of the box-spline basis as the 0^{th} order b-spline is the box-spline.
- By varying the knots (basically the "edge of a bin" of a box-spline) we can change the resolution of the b-splines.

For details on b-splines, see Ref. [21] and for detail on how we use it in the inversion problem, see Ref. [3]. The actual implementation that we use is CERNLIB's NORBAS package, E210 [22].

In the b-spline basis, the concept of the "edge of a bin" in the box-spline basis is replaced with the concept of a knot. A knot is simply the place where the polynomials that make up the b-spline are patched together. In the "optimized discretization" scheme of Ref. [2], the edges of the box-splines are varied to minimize the relative error of the source. We may generalize this idea to the b-splines easily by varying the locations of the knots.

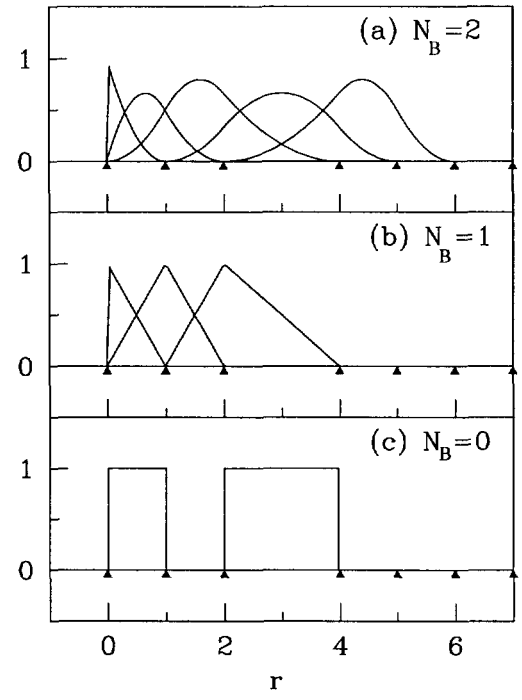

Figure 1.2: Sample plots of N_B^{th} degree b-splines. In all panels, the knots are marked by carets and the knots at $r = 0$ are actually $N_B + 1$ regular knots piled together.

1.2.5 Optimal Knots for the B-splines

Since the kernel is not truly a Fourier Transform, we must ask whether the finite sized binning implied by Fourier theory is optimal. Also owing to various experimental effects, we may be insensitive to various regions in r space and may wish to lower the resolution in these places. To deal with these two problems, we use the generalization the "optimized discretization" scheme of Ref. [2] detailed in Ref. [3].

First, we notice that the model covariance matrix of Eq. (1.20) depends on the kernel of the inversion, the error on the data and whatever scheme we use to represent the source, *but not on the correlation data or the source itself.* For a given kernel and set of data errors, we are free to change our representation of the source in order to minimize the error of the source. In particular, we may vary the location of the knots (at least not the knots fixed at the endpoints of the imaging region) to minimize the error of the source, $\Delta S_j = \sqrt{\Delta^2 S_{jj}}$, relative to some dummy source:

$$\sum_{j=1}^{N_S} \left| \frac{\Delta S_j}{S_j^{dummy}} \right| = \text{min.} \qquad (1.22)$$

The coefficients S_j^{dummy} are the expansion of a dummy source in b-splines. In this minimization, the first and last multiple knots are held fixed and the positions of all of the other knots are varied. The dummy source itself is chosen to be big roughly where one expects the source to be big and small where one expects the source to be small. Since the details of the dummy source are not important, we choose a dummy source to be an exponential with radius $R^{dummy} = 3.5$ fm given by $S^{dummy}(r) \propto \exp(-r/R^{dummy})$.

1.2.6 Constraints

In order to further stabilize the inversion, we can take advantage of prior information in the form of constraints, as first suggested by Tikhonov [23]. Here we focus on equality constraints. An equality constraint is a condition on the vector of source coefficients that has the generic form $\mathcal{C} \cdot \mathbf{S} = \mathbf{c}$. One example of such a constraint is that the source has slope 0 at the origin, in which case we write

$$S'(r \to 0) = \sum_{i=1}^{N_S} S_i B_i'(r \to 0) = 0. \qquad (1.23)$$

Other useful constraints are detailed in Section 5.2.

We can implement these types of constraints by just adding a penalty term to the χ^2:

$$\chi^2 + \lambda(\mathcal{C} \cdot \mathbf{S} - \mathbf{c})^2 \tag{1.24}$$

Here the λ is a trade-off parameter and we may vary it in order to emphasize stability in the inversion (by making λ huge) or to emphasize goodness-of-fit (by setting λ to zero). Such an ability to trade-off stability for goodness-of-fit is discussed in Numerical Recipes [24] in detail. With this modification of the χ^2, the imaged source is

$$\mathbf{S} = \Delta^2 S \cdot \left(K^T \cdot (\Delta^2 \mathcal{R})^{-1} \cdot \mathcal{R}^{\mathbf{obs}} + \lambda \mathcal{C}^T \cdot \mathbf{c}\right), \tag{1.25}$$

and the covariance matrix of source now is

$$\Delta^2 S = \left(K^T \cdot (\Delta^2 \mathcal{R})^{-1} \cdot K + \lambda \mathcal{C}^T \cdot \mathcal{C}\right)^{-1}. \tag{1.26}$$

This technique may be justified using Bayes Theorem as explained in Refs [3, 25].

As an alternative to this approach, constraints may be implemented exactly using the Householder Reduction of a matrix composed of the constraints and the inversion kernel together. Such a scheme is explained in more detail in Numerical Recipes [24] as well as in CERNLIB's documentation for the TL package, no. E230 [22].

Inequality constraints may also be used to stabilize the inversion. Unfortunately, implementing them in the code requires the use of so-call "Active set" methods, which are beyond our current ability. Nonetheless, we do check to see how well inequality constraints are obeyed by the images.

Chapter 2

GETTING STARTED

2.1 Obtaining the Source Code

To obtain the source codes for these programs, you can contact either one of the authors, Dave Brown (at **brown170@llnl.gov**) or Paweł Danielewicz (at **danielewicz@nscl.msu.edu**), or download the codes from LLNL's Theory and Modeling Group web-page at `http://www-phys.llnl.edu/Organization/NDivision/ntm/HBTprogs.html`. The codes are distributed as a Unix tarball (`HBTprogs.tar.gz`) or a zipfile (`HBTprogs.zip`).

2.2 System Requirements

We have test the codes on several different systems:

- RedHat Linux 7.1 using g77 v0.5.26 from Gnu Compiler Collection, gcc v2.96-85.
- Compaq Tru64 using f90 and f77, the Compaq Fortran Compiler V5.4A-1472-46B2F.

It probably works on many other systems, so if you get it to work on another system, please let us know and we'll add it to this list.

In addition to the Fortran compiler, you will also need several other things:

- `tar` and `gzip` to unpack the tarball
- LaTeX and/or a postscript viewer to read the manual.
- `make` to build the programs
- `sh` to run the shell scripts
- `GNU autotools` for some of the build features in the `configure` script (`autoconf` version 2.52f and `automake` version 1.5 were used to build this script).
- A text editor
- CERNLIB libraries (the 1997a, 1999, and 2000 versions seem to work).
- a FORTRAN compiler
- A c++ compiler if you want to use `mtx2xpm.cpp` or the `CRAB` add-on programs
- `CRAB`, for the `CRAB` add-on programs

2.3 Installation

The installation is fairly simple under Unix:

1. Set the environment variable `CERNLIB` to the path to the CERNLIB library files `libpacklib.a` and `libmathlib.a`
2. Unpack the tarball:

```
% gzip -d HBTprogs-v1.0.tar.gz
% tar xvf HBTprogs-v1.0.tar
```

3. Go to the `HBTprogs-v1.0` directory

```
% cd HBTprogs-v1.0
```

4. Run the `configure` script:

```
% ./configure
loading cache ./config.cache
checking for a BSD compatible install... (cached) /usr/bin/install -c
checking whether build environment is sane... yes
checking whether make sets ${MAKE}... (cached) yes
checking for working aclocal... found
checking for working autoconf... found
checking for working automake... found
...
creating src/Makefile
creating src/kernel-generators/Makefile
creating src/1d/Makefile
creating doc/Makefile
```

5. Help may be obtained by typing "configure –help"

6. Make the manual, library, etc.

```
% make install-data
Making install-data in lib
make[1]: Entering directory '/home/dbrown/Projects/Interferometry/HBTprogs-v1.0/lib'
make[1]: Nothing to be done for 'install-data'.
make[1]: Leaving directory '/home/dbrown/Projects/Interferometry/HBTprogs-v1.0/lib'
Making install-data in src
make[1]: Entering directory '/home/dbrown/Projects/Interferometry/HBTprogs-v1.0/src'
Making install-data in kernel-generators
make[2]: Entering directory '/home/dbrown/Projects/Interferometry/HBTprogs-v1.0/src/'
/bin/sh ../../config/mkinstalldirs ../../kernels
  /usr/bin/install -c  genmaNI1D ../../kernels/genmaNI1D
  /usr/bin/install -c  genmaPP1D ../../kernels/genmaPP1D
  /usr/bin/install -c  genmaIMF1D ../../kernels/genmaIMF1D
  /usr/bin/install -c  genmaKK1D ../../kernels/genmaKK1D
  /usr/bin/install -c  genmaPI1D ../../kernels/genmaPI1D
...
```

That's it!

Chapter 3

An Example Run

In this section, we describe how to run the codes. Since it is easiest to image if you use the `imageit` script, we will concentrate on using it. We begin by performing the example imaging calculation in the the `doc/example/` directory and a detailed description of the `imageit` script is contained in Chap. 5.

This example is a simple one in that we will start with the proton correlation contained in `example_pp.corin`, corresponding to the Gaussian source function in `example_pp.souin`. This Gaussian has $\lambda = 1$ and $R = 5$ fm.

1. If you already have not done so, move to the `doc/example/` subdirectory.

2. Copy the `imageit` script from the `sbin/` directory and change its permissions so that it is executable.

3. We need to figure out how many points to use the reconstruction, so view the `example_pp.corin` file and count the number of bins where $C(q) \neq 1$. In this example, there should be 15 bins.

4. Now edit `example_pp.dimensions`. First, change `Ndata` (this value is N_D in the equations in the text) to correspond to the total number of data points in `example_pp.corin`. Second, change `Nmodel` to be some number smaller than the number of bins you found in the previous step (`Nmodel` is N_M, the number of source coefficients in the text). `Nmodel`= 8 bins should work .

5. Edit `example_pp.bsplines` and set the order of the b-splines, `BSplineDegree`. `BSplineDegree`= 3 should work.

6. For the last bit or preparation, edit the `example.controls`. We have several things to edit here. As you can see from the file, the controls are separated into a set of six different FORTRAN namelists. We will divide our explanation into these six parts.

   (a) The `data_format` namelist: these controls describe the format of `example_pp.corin`. Since there are only three columns in the file, set `xydxdy=.false.` and set `bigQ=.false.` since $q = q_{inv} = \frac{1}{2}\sqrt{-(p_1 - p_2)^2}$, not $Q_{inv} = 2q_{inv}$. Also, since the data is already in GeV/c, set `energy_factor=1.d0`.

   (b) The `knot_controls` namelist: these controls effect how the knots are chosen and displayed. They are described in detail later. For now, set `show_knots=.false.`, `save_knots=.true.`, and `knotmode=1`.

   (c) The `fit_controls` namelist: these control the various fitting controls. For simplicity, set all constraints to `.false.`, turning them off. Set the fit limits to be `low_fit_lim=0.d0` and `up_fit_lim=45.d0`. Since we have turned off all constraints, the `tradeoff` parameter is irrelevant.

   (d) The `norm_constraint_controls` namelist: these controls are for the normalization constraint. Since we turned it off, this section does nothing.

   (e) The `display_controls` namelist: these controls determine what is information is outputted from the inversion. The image file, coefficient file and covariance matrix file are always created, so for simplicity

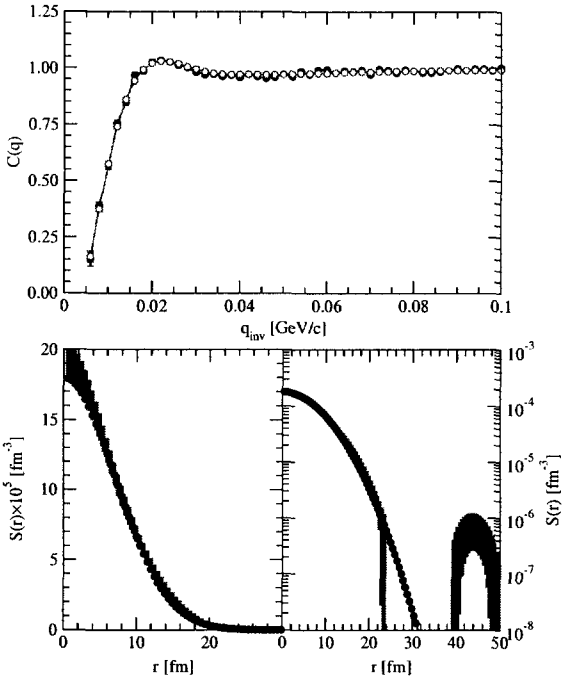

Figure 3.1: Results of the example calculation. The input model correlation and source are shown in black and the restored source and correlation are shown in red. The restored source is represented as an error band. The two lower panels show the source on both a linear and log scale.

set all the options to `.false.`. You may set `up_int_lim` to any positive number less than `up_fit_lim`. The code will integrate the imaged source up to that value.

(f) The `special_3D_controls` namelist: these controls are specifically for the 3d inversion code that is still in development.

7. Now we are ready to run the code. Invoke `imageit`:

```
% ./imageit example pp scratch
```

Here, `scratch` is a directory for the codes to use to store temporary files.

8. That's it!

In the `doc/example/` directory, you should find several new files. The most important ones are the files `example_pp.coeffs`, which contains the coefficients of the b-spline expansion of the source, `example_pp.covmtx`, which contains the covariance matrix of the imaged coefficients, and `example_pp.knotlist`, which contains the knots characterizing the b-spline expansion. Unfortunately, none of these files gives you a simple view of the source. For that, you should plot the `example_pp.imag` which is the source tabulated as $S(r)$ vs. r. These results are plotted in Fig. 3.1.

Chapter 4

Preparing Input Files

This chapter details the various input files needed for the codes. We begin with a discussion of the file-naming conventions. Next, we explain the format for the correlation data. Following this, we describe the FORTRAN include files that specify the array sizes. Finally, we list and explain all of the options in the main control file.

4.1 Filename conventions

There are many input files for the imaging codes, and there are even more output files. This is a reflection of our use of FORTRAN as well as our programming ability. Ideally we would like all inputs to be in one big file, with clean syntax. Since the user must be able to change array sizes and FORTRAN lacks dynamic memory allocation, every time the one big file is changed, the user would need to recompile the code. Since other information can be loaded after compilation, we instead use a few different files.

The file names for these files are hard-coded into the codes. Since the management of these files can be somewhat complicated, we wrote a few scripts to rename the files to/from these hard-coded names to more reasonable names. The scheme we chose is simple: all files in a run have the same prefix and different extensions, e.g. `yourfile.extension`. A listing of all of the extensions is shown in Table 4.1 The mapping of user filenames to working filenames is shown in Table 4.2. There is a similar naming convention for the `makco1D` code and this is discussed in Chap. 7.

4.2 Correlation data file format

The correlation function data should be placed in the file `yourfile.corin`. There are a few variations on the input format that the codes can use and here we will quickly outline them. You may use either 3 or 4 columns to input the correlation (controlled by the `xydxdy` flag in `yourfile.controls` file. Additionally, you have nearly complete control over the input units for the relative momentum. In all cases, we assume that the center of the i^{th} bin is q_i and the width of i^{th} the bin is Δq_i (so that the i^{th} bin extends from $q_i - \Delta q_i/2$ to $q_i + \Delta q_i/2$).

4.2.1 3 Column Format

The 3 column format contains, oddly enough, 3 columns of real numbers. The first column is the q of the bin, the second column is the correlation function itself, and the final column is the error on the correlation function. If you use this format, the code will assume that all of the Δq_i's are equal.

As an example, the sample `yourfile.corin` file look like:

```
4.000  0.3413868  4.1696317E-02
6.000  0.5828804  3.9426915E-02
8.000  0.7180706  3.0471841E-02
...
```

The Fortran statement that reads this file is

```
READ(30,*)Q(ISQ),C(ISQ),dC(ISQ)
```

Note that the momentum here is specified in units of MeV/c, although nearly any units are acceptable.

We recommend that you use this option if your bins all have the same width.

4.2.2 4 Column Format

The 4 column format adds a column corresponding to `dq` $= \Delta q/2$, the error on the momentum (in other

Input file extensions	
extension	file description
`.bsplines`	FORTRAN include file that specifies the order of the b-spline expansion.
`.controls`	Main control file.
`.corin`	Correlation data.
`.dimensions`	FORTRAN include file that specifies various array sizes.
`.knots`*	List of knots (needed if `knotmode`=4).

Output file extensions	
extension	file description
`.coeffs`	Final coefficients of the source after fitting.
`.covmtx`	Covariance matrix of the source coefficients.
`.imag`	A table of $S(r)$ vs. r suitable for plotting.
`.knotlog`	Log file from the knot determination (if `knotmode`=3).
`.knots`	List of knots used for the b-splines.
`.optres`	Resolution matrix. This is not explained in the text.
`.restored`	The inverted-then-uninverted correlation for use as a cross check.

*used only for `knotmode=4`, see section 5.1 for more detail on choosing the knots

Table 4.1: List of the file extensions used by the `imageit` script.

Input Files	
starting name	working name
`yourfile.corin`	`corinput.dat`
`yourfile.dimensions`	`dimensions1D.inc`
`yourfile.bsplines`	`bsplines1D.inc`
`yourfile.controls`	`main.controls`
`yourfile.knots`	`knotlist.dat`

Output Files	
working name	ending name
`source-imag.dat`	`yourfile.imag`
`source-coeffs.dat`	`yourfile.coeffs`
`source-covmtx.dat`	`yourfile.covmtx`
`knotlist.dat`	`yourfile.knots`
`makeknots.log`	`yourfile.knotlog`
`c2_model.dat`	`yourfile.restored`
`optresmtx.dat`	`yourfile.optres`

Table 4.2: Mapping of starting filenames to working filenames and back to ending filenames.

words, the $\frac{1}{2}$-width of each bin). This column is added before the error on the correlation column. The above sample file would then be:

```
4.000  0.3413868  2.000  4.1696317E-02
6.000  0.5828804  2.000  3.9426915E-02
8.000  0.7180706  2.000  3.0471841E-02
...
```

The Fortran statement that reads this file is

```
READ(30,*)Q(ISQ),C(ISQ),dQ(ISQ),dC(ISQ)
```

If your bins do not all have the same size, *you must use this format to specify the bin widths.*

4.2.3 Units of relative momentum

The q that is entered in the table may be in *any* units and it may correspond to $q_{inv} = \sqrt{-(p_1 - p_2)^2}/2$ (`bigQ=.false.`) or $Q_{inv} = \sqrt{-(p_1 - p_2)^2}$ (`bigQ=.true.`). Whatever units you choose, you must specify how to convert your choice of units to our choice of units in `yourfile.controls`. How exactly you are to do this is specified in the section on the `yourfile.controls` file, but basically you must specify the units conversion factor and say whether you will use q_{inv} or Q_{inv}. In any event, inside the codes we work in GeV/c and our default q is q_{inv} for both all particle pairs (including mesons where one traditionally uses Q_{inv}).

4.3 Editable include files

Files `yourfile.bsplines` and `yourfile.dimensions` are files that get included by `makco1D` and `image1D` when they are compiled. Setting up `yourfile.bsplines` is easy. Once you have chosen what order b-splines you want to use (0 for box-splines, 1 for hat functions, ...), simply change the line

```
parameter (BSplineDegree=3)
```

to reflect that order.

Setting up `yourfile.dimensions` may take a little more thought. Here the line to edit is

```
parameter (Ndata=83,Nmodel=6,Neqcon=5)
```

First, set `Ndata` equal to the number of bins in your data file. Choosing `Nmodel` is a little more difficult. To get a rough guess, count up the number of data points in you correlation that look like they are different from 1. Let's call this `Ninteresting`. Now choose how many equality constraints you think you might use and call that `Nusedcons`. With these, a starting guess for `Nmodel` is:

$$\texttt{Nmodel} \approx \texttt{Ninteresting} + \texttt{Nusedcons} \qquad (4.1)$$

Chapter 5 and Ref. [3] discuss other ways of setting this variable. The last variable `Neqcon` sets the maximum number of constraints in the code. This should not be changed unless you plan on adding more equality constraints.

4.4 Main control file

The main control file is `yourfile.controls`, which consists of a series of FORTRAN namelists. Each namelist corresponds to a specific set of options and here we detail each namelist separately.

- **The `data_format` Namelist.** This namelist contains the options that specify the format of your input correlation function, `yourfile.corin`. The options are:

  `xydxdy`: This sets whether the data file has either a four column format, e.g. $q_i, C(q_i), dq_i, dC(q_i)$ (`xydxdy = .true.`), or a three column format, e.g. $q_i, C(q_i), dC(q_i)$ (`xydxdy = .false.`).

  `bigQ`: This sets whether q corresponds to $q_{inv} = \sqrt{-(p_1 - p_2)^2}/2$ (`bigQ = .false.`) or to $Q_{inv} = \sqrt{-(p_1 - p_2)^2}$ (`bigQ = .true.`).

  `energy_factor`: This tells the codes how to rescale the q's in `yourfile.corin` get rescaled to bring them to units of GeV/c (the internal format of all of the codes). For example, if your data is in MeV, then `energy_factor=1d.-3`

- **The `knot_controls` Namelist.** These options tell the imaging code what to do with the knots.

  `show_knots`: This tells `image1D` to print the knot list to the screen.

  `save_knots`: This tells `image1D` whether to save the knot list to the file `knotlist.dat`.

knotmode: This option tells image1D which method to use to chose the knots. The options are descibed in detail in Section 5.1.

- **The fit_controls Namelist.** These options control the actual imaging of the source.

  low_fit_lim: This option sets the lower bound of the region in r that you want to reconstruct the source over.

  up_fit_lim: This option sets the upper bound of the imaging region.

  tradeoff: If tradeoff is positive, it is the trade-off parameter λ in Eq. (1.24). It should be chosen so that the constraint term in the modified χ^2 is larger than the data term. According to Ref. [3], a good choice is to pick tradeoff$\gg$ (Ndata+Nmodel+1)$\times 10^6$ If tradeoff is negative, this signals to image1D to use a Householder reduction to satisfy the constraints exactly. One should note however that it may not be possible to both satisfy the constraints exactly and obtain a reasonable image.

  diff_constraint: This option tells image1D whether to use the $S'(r \to 0) = 0$ constraint.

  norm_constraint: This option tells image1D whether to constrain the integral of the source. See below for more options for this constraint. (Note: this constraint does not work as of 8/8/2000, DAB).

  zero_source_at_large_r: This option tells image1D whether to force $S(r) = 0$ when r =up_fit_lim.

  smooth_source_at_large_r: This option tells image1D whether to force $S'(r) = 0$ when r =up_fit_lim -5 fm. This has the effect of smoothing the source at large distances. This option sometimes has no effect when used in conjunction with zero_source_at_large_r.

- **The norm_constraint_controls Namelist.** This namelist contains more fine-grained control over the normalization constraint. (Note: these options do not work as of 8/8/2000, DAB)

  norm: This sets what value you want the norm of the source constrained to. The default is norm = 1.d0.

  norm_int_lim: This sets the upper limit of the integral used to determine the normalization of the source.

- **The display_controls Namelist.**

  show_source: This option tells image1D whether to write the imaged source out to the screen. It is always written to the file source-imag.dat.

  show_coeffs: This option tells image1D whether to write the source coefficients to the screen. It is always written to the file source-coeffs.dat.

  show_corr: This option tells image1D whether to uninvert the source to obtain a correlation that may be compared to your input correlation. The results are written to the screen and to the file c2_model.dat.

  up_int_lim: image1D always integrates the source and outputs the results. The upper limit of this integration is up_int_lim.

  check_ne_constraints: This option tells image1D whether to check if the imaged source obeys the inequality constraints and then outputs the results.

  show_resmtx: This is an undocumented feature.

  show_kernel: This option tells image1D whether to write the kernel it used to invert the correlation to the screen.

- **The special_3D_controls Namelist.** This namelist contains the set of options specific to the three dimensional inversion code which is currently in development. Thus, we do not discuss it at this time.

For more information on setting these files, see Chapter 5 on imaging or the FAQ (Chapter 8).

Chapter 5

IMAGING THE SOURCE

This chapter details how to use the `image1D` code and the wrapper script `imageit` to invert correlation data. To invoke `imageit`, type

```
$ ./imageit [prefix] [PID] [scratchdir]
```

The `[prefix]` argument of `imageit` is the prefix of your correlation function. The `[PID]` argument is the particle type. Valid types are shown in Table 6.1. The `[scratchdir]` argument is an optional directory to store the temporary files.

While the `imageit` script simplifies the running of the imaging code, to use the code effectively you must understand the various options. Most are adequately explained in Chap. 4. Here we focus on choosing the knots, the use of equality constraints and checking whether inequality constraints are obeyed.

5.1 Choosing the knots

The knots are the matching points of the polynomials comprising the b-splines. Because the width of the b-splines set the resolution of the image, picking good knots is crucial. Unfortunately, choosing the knots is not always straightforward. In this section, we will explain how to choose the knots and explain the various things you may do to influence the location of the knots. The most obvious way to set the knots is by picking a `knotmode`. Table 5.1 lists the various `knotmode`'s available to you. While Table 5.1 may be enough information to use the code, there are many subtle points that we must elaborate on in this section.

The outline of this section is as follows. First, we discuss some general issues with our knot choices. Second, we detail the special case of box-splines, where the knots are actually the edges of the bins of the histogram that the box-splines represent. Finally, we detail the individual knot modes that you may choose from. If you would like more detail on the topic of choosing the knots, you should probably read Refs. [3, 21].

5.1.1 General comments about knots

In this subsection, we briefly describe how we determine the number of knots in an expansion and how we fix the first and last few knots.

The knots form an ordered list, $\{t_i\}$, of N_K values on the interval [`low_fit_lim`, `up_fit_lim`]: `low_fit_lim` $\leq t_1 \leq t_2 \leq ... \leq t_{N_K} \leq$ `up_fit_lim`. The number of knots is $N_K = N_M + N_B + 1$, where N_M is the number of source coefficients (`Nmodel` in the code) and N_B is the degree of the spline representation (`BSplineDegree` in the code).

Since we do not know the behavior of the source at large r and we truncate the source long before it completely goes to 0, we do not make any continuity assumptions at `up_fit_lim`. We do know that the source should go to a maximum at $r = 0$ fm, however the peak at the origin may be too sharp to resolve. Thus, we can not always make continuity assumptions here either. In the end, we must leave the control of the continuity of the source at the edges of the fit region to the user. In order to remove any assumptions about the degree of continuity at the edges of the fit interval, we fix both the first and last N_B+1 knots at their respective fit interval edge.

5.1.2 Box-splines

In this subsection, we describe the special case of the box-spline. This is the case where the `BSplineDegree`=0 and is more commonly called box-

knotmode	routine used	description	notes
1	CERNLIB's DSPKN1	fixed spaced knots	
2	CERNLIB's DSPIN1	set knots to collocation points for Schoenberg's variation diminishing approximation	this option overrides up_fit_lim
3	makeknots2	"optimal knots"	this option requires CERNLIB's minuit, so may not function on some linux machines
4	n/a	user supplied	requires knotlist.dat

Table 5.1: Various settings for knotmode. All of the CERNLIB routines are detailed in the CERNLIB documentation [22].

splines. This case must be describe separately as it has several properties that you must be aware of when using them. We define box-splines here and comment on their continuity properties.

For a given list of knots $\{t_i\}$, the i^{th} box-spline is

$$B_i(r) = \theta(r - t_i) - \theta(r - t_{i+1}). \quad (5.1)$$

Thus, the knots that define the b-splines in this case are just the edges of the bins of the box-splines. Expanding a function in terms of box-splines results in a histogramed representation of the function. Some box-splines are pictured in Fig. 1.2(c).

A function expanded in the box-spline basis clearly has discontinuities at all bin boundaries. Thus when we use this basis to represent the source, we are making no assumptions about the degree of continuity of the source. In fact, setting BSplineDegree=0 deactivates all constraints on the source that deal with its continuity.

5.1.3 Specific knotmode's

We now detail the various knotmode's. For most practical purposes, the equally spaced knots (knotmode=1) should be good enough, however we provide other options for more control over the knot placement.

Equally spaced knots

This is the simplest knot choice and the best option for following the Fourier theory recommendations in Sect. 1.2.3. In practice, we have found this option to be a good default choice. Under this option, the first and last BSplineDegree+1 knots are fixed at the fit region limits. The remainder of the knots are equally spaced between these end knots.

Schoenberg's Variation Diminishing Spline knots

Schoenberg's Variation Diminishing Spline Approximation is a scheme for approximating a function using b-splines. The goal of the approximation is to minimize the difference between the b-spline approximation and the true function in a least-square sense [21, 22]. In order to make the approximation work, special knots must be chosen. We do not use this approximation, but we were interested in experimenting with the knot choice used in this approximation.

In this approximation, the knots are built off a list of control points, $\{x_i\}$, via

$$x_i = (t_{i+1} + ...t_{i+N_B})/N_B. \quad (5.2)$$

The control points are chosen such that the first and last control points are set to the end of the fit region and the rest of the control points are equally spaced between them. Unfortunately, Eq. 5.2 is not uniquely invertible. Ref [21] offers a prescription that is used in CERNLIB. This prescription is basically an iterative method that results in knots $\{t_i\}$ that approximately satisfy Eq. (5.2).

Optimal knots

This option is already adequately explained in Sect. 1.2.5.

Do-it Yourself Knots

This option is somewhat of a last resort for when no other `knotmode` seems to work. To use this, it is probably best to start with a `yourfile.knots` file generated using one of the other options. To generate a valid (although not necessarily good) knot list, you must do the following:

- There should be `Nmodel+BSplineDegree+1` knots.
- The first `BSplineDegree+1` knots should be fixed to `low_fit_lim` and the last `BSplineDegree+1` knots should be fixed to `up_fit_lim`.
- The knots must be monotonically increasing.

While we use this option rarely, we have found cases where noticeable improvements in the final χ^2 may be found by moving the knots just a little bit.

5.2 Equality Constraints

As we have explained earlier, an equality constraint is a condition on the source that can be written $C \cdot \mathbf{S} = \mathbf{c}$. There are several constraints that we find useful and they are summarized in Table 5.2. They each arise from known physics of the two particle source, and they each serve a mathematical purpose. In Table 5.3, we also list the equations of the constraints and their b-spline representation.

The degree to which the constraints are obeyed is set by the `tradeoff` parameter. This parameter is the λ parameter mentioned in Eq. (1.24). If `tradeoff` < 0, this signals the code to use the Householder reduction to exactly obey the equality constraint. This is often a good choice, but it may happen that your source (or your representation of the source) is incompatible with the constraints. This sometimes happens with very narrow sources when using the `diff_constraint` option. If `tradeoff` > 0, this signals the code to function as described in (1.25). The limit where `tradeoff` $= \lambda \to \infty$ exactly reduces to the Householder reduction case.

As a final comment to using equality constraints, we mention that you should consider increasing the number of source coefficients so as to avoid overconstraining the system and producing garbage. The rule of thumb we have found has been stated already in Eq. (4.1).

5.3 Inequality Constraints

As we have mentioned earlier, an inequality constraint is a constraint on the source of the form $C \cdot \mathbf{S} \geq \mathbf{c}$. We have not implemented inequality constraints in the inversion. However, we do check to see if any are violated.

We check the following inequality constraints for compliance:

`positivity`: Since the source is a convolution of two quantum mechanical Wigner functions, it is possible that it could go < 0. However, classically, the phase space density is positive definite, so the source must also be ≤ 0. In our experience, a negative source indicates a problem with the imaging.

`FT test`: If the independent source hypothesis is correct, then according to Ref. [16] the Fourier transform of the source must be positive definate. Sources that fail this test also tend to have problems with the imaging.

`normalization`: Conservation of probability implies that the integral of the source over all r should yield 1. However, if one truncates the integral at say `up_fit_lim`, then the integral should be ≤ 1. This can help diagnose wild fluctuations in the imaged source.

A list of these inequality constraints, along with their b-spline representations, are in 5.4.

constraint	justification	purpose
`diff_constraint`	$S(\mathbf{r})$ is convolution of two well behaved single particle emission rates so $r = 0$ fm is a maximum.	To constrain the higher ℓ components that are not well controlled due to the effect of the centrifugal barrier in the Schrödinger equation on the pair wavefunction.
`norm_constraint`	Conservation of probability forces source to be normalized to 1.	To smooth fluctuations at large r, which dominate the normalization integral.
`zero_source_at_large_r`	No pairs are created farther than $\|\mathbf{r}\| = r_{max}$ apart in the model.	To smooth oscillations in the source at high $\vec{r}$ caused by aliasing of statistical and experimental noise in correlation.
`smooth_source_at_large_r`	$S(\mathbf{r})$ should go to 0 smoothly as $r \to r_{max}$.	Same as `zero_source_at_large_r`.

Table 5.2: Explanation of the various equality constraints.

constraint	continuous representation	b-spline representation
`diff_constraint`	$\frac{\partial S}{\partial r}(r \to 0) = 0$	$\sum_{j=1}^{N_M} S_j B_j'(r \to 0) = 0$
`norm_constraint`	$4\pi \int_0^\infty dr\, r^2 S(r) = \lambda$	$\sum_{j=1}^{N_M} 4\pi S_j \int_0^\infty dr\, r^2 B_j(r) = \lambda$
`zero_source_at_large_r`	$S(r_{max}) = 0$	$\sum_{j=1}^{N_M} S_j B_j(r_{max}) = 0$
`smooth_source_at_large_r`	$\frac{\partial S}{\partial r}(r_{max}) = 0$	$\sum_{j=1}^{N_M} S_j B_j'(r_{max}) = 0$

Table 5.3: Equality constraints on the b-spline representation of spherically symmetric sources.

constraint	continuous representation	b-spline representation
positivity	$S(r) \geq 0$	$\sum_{j=1}^{N_M} S_j B_j(r) \geq 0$
FT test	$4\pi \int_0^\infty dr\, r^2 \frac{\sin(2qr)}{2qr} S(r) \geq 0$	$\sum_{j=1}^{N_M} 4\pi S_j \int_0^\infty dr\, r^2 \frac{\sin(2qr)}{2qr} B_j(r) \geq 0$
normalization	$4\pi \int_0^\infty dr\, r^2 S(r) \leq 1$	$\sum_{j=1}^{N_M} 4\pi S_j \int_0^\infty dr\, r^2 B_j(r) \leq 1$

Table 5.4: Inequality constraints on the b-spline representation of spherically symmetric sources.

Chapter 6

BUILDING KERNELS

This section focuses on building the kernels and how to construct your own kernels. In general, making a kernel file is simple using the `pick-kernel` shell script:

```
% ./pick-kernel [PID] [link]
```

Here, the `[PID]` option is actually the keyword to tell `pick-kernel` which kernel to build. A table showing the valid kernel keywords is shown in Table 6.1. The `[link]` option tells `pick-kernel` where you would like to symlink the created kernel.

The `pick-kernel` script is used by the two control scripts (`imageit` and `createtest`) to choose the kernel. Since `pick-kernel` simply manages the production of the kernels, the actual work is done by the various `genma*1D` codes. While using `pick-kernel` and the underlying kernel generators is simple, one should still know how they work, both to understand the pitfalls and limitations of each type of pair, and to understand how to write one's own kernel code for pairs we had not considered.

We now outline this section. First, we detail the `pick-kernel`. Second, taking the non-interacting boson kernel as an example, we explain the layout of the kernel files. This should help you create your own kernel. Next, we explain the proton-proton, IMF and meson kernels. Finally, we explain how the kernels are loaded and used in `image1D`.

6.1 The `pick-kernel` script

If you ever make a new kernel, you should consider modifying the `pick-kernel` script to automate the use of you kernel. So, to aid in this, we now explain briefely how the script works. You should also examine the source itself though.

Here is an outline of the script:

1. Map keyword `[PID]` to kernel generator and output data file. See Table 6.1.
2. Go to `kernels/` directory in source tree.
3. See if the requested kernel is already done. If so, skip step 4.
4. Build the kernel and save it as `genma*.dat`.
5. Make a symbolic link in from kernel in `kernels/` to the final destination, `[link]/kernel.dat`.

6.2 Layout of `kernel.dat` and `genmaNI1D`

This section is a explains how the `kernel.dat` files are layed out, so that you can troubleshoot existing kernels or, perhaps, create your own kernel generator.

For non-interacting bosons, the final state relative wavefunction is simply a symmetrized plane wave:

$$\Phi_{\mathbf{q}}(\mathbf{r}) = \frac{1}{\sqrt{2}}\left(e^{i\mathbf{q}\cdot\mathbf{r}/\hbar c} + e^{-i\mathbf{q}\cdot\mathbf{r}/\hbar c}\right). \qquad (6.1)$$

resulting in a rather trivial kernel: $K(\mathbf{q},\mathbf{r}) = \cos(2\mathbf{q}\cdot\mathbf{r}/\hbar c)$. It is easy to show that the angle averaged kernel is therefore:

$$K_0(q,r) = \frac{\sin\rho}{\rho} \quad \text{where} \quad \rho = 2qr/\hbar c. \qquad (6.2)$$

Note, the kernel has no units. The simple FORTRAN code that writes this kernel to `genmaNI1D.dat` is listed in Fig. 6.1.

Although one may read the file format off this FORTRAN code, it does not hurt to repeat it here.

```
c-----------------------------------------------------------------------
c     genmaNI1D.f -- generates 1D inversion matrix assuming
c                    BE statistics and no FSI
c-----------------------------------------------------------------------
c     program genmaNI1D
      implicit none
      include 'constants.inc'   !fund. consts. (hbar, pi, etc...)
      include 'particle.inc'    !particle masses & charges
      include 'gridinfo1D.inc'  !define r & q grid here
      real*8 Kmatrix(nsr)
      real*8 r,qrel, rho
      integer isq,isr
c
c     open kernel file and create header
c
      write(*,*) 'Writing kernel to disk'
      open(20,file='genmaNI1D.dat',status='unknown')
      write(20,*) 'Generated by genmaNI1D'
      write(20,*) 'Z1*Z2= 0'
      write(20,*) 'reduced_mass= 0'
      write(20,*) 'Nsq= ',nsq
      write(20,*) 'Nsr= ',nsr
      write(20,*) 'maximum_Lambda= 0'
      write(20,*) '__begin_table__'
      write(20,*) '  Lambda= 0'
c
c     find & output complete K-matrix
c
      do isq=1,nsq                !start big q loop
         qrel=dble(isq)*dsq+qmin
         write(*,'(1H+,I3)')isq
         do isr=1,nsr             !start r loop
            r=dble(isr)*dsr+rmin
            rho=2.d0*qrel*r/hc
            if (rho.lt.1.d-6) then
               Kmatrix(isr)=1.d0-rho*rho/6.d0
            else
               Kmatrix(isr)=sin(rho)/rho
            endif
         enddo                    !end r loop
         write(20,*)(Kmatrix(isr),isr=1,nsr)
      enddo                       !end big q loop
c
c     output footer and close file
c
      write(20,*) '__end_table__'
      close(20)
      end
c-----------------------------------------------------------------------
```

Figure 6.1: Listing of the program genmaNI1D.f.

pair	keyword
$\pi^+\pi^+$	pi+pi+
$\pi^-\pi^-$	pi-pi-
K^+K^+	K+K+
K^-K^-	K-K-

pair	keyword
no FSI	NI
pp	pp
IMF's	IMF

Table 6.1: The keyword to particle pair mapping used by `pick-kernel`.

```
Generated by genmaNI1D
Z1*Z2= 0
reduced_mass= 0
Nsq=  1001
Nsr=  201
maximum_Lambda= 0
__begin_table__
 K0(q1,r1)    K0(q1,r2)    ...   K0(qNsq,r1)
 K0(q2,r1)    K0(q2,r2)    ...   K0(qNsq,r2)
    ⋮            ⋮          ⋱        ⋮
K0(qNsq,r1)  K0(qNsq,r2)  ...  K0(qNsq,rNsr)
__end_table__
```

Figure 6.2: Layout of the kernel file produced by `genmaNI1D.f`.

A schematic version of `genmaNI1D.f` is shown in Figure 6.2.

The grid itself is specified in `gridinfo1D.inc` and here are the equations for the grid points:

$$\begin{aligned} q_{\mathtt{isq}} &= \mathtt{dble(isq)} * \mathtt{dsq} + \mathtt{qmin}, \\ r_{\mathtt{isr}} &= \mathtt{dble(isr)} * \mathtt{dsr} + \mathtt{rmin}. \end{aligned} \tag{6.3}$$

6.3 The rest of the kernels

In this section, we detail the rest of the kernels, commenting on some of the trickier aspects of the codes.

6.3.1 genmaPP1D, for proton pairs

The proton kernel is given by the following sum over partial waves:

$$K_0(q,r) = \frac{1}{2}\sum_{js\ell\ell'}(2j+1)(g_{js}^{\ell\ell'}(r))^2 - 1. \tag{6.4}$$

The sums over spin s, total angular momentum j and orbital angular momenta ℓ and ℓ' basically amount to an overall factor of $\frac{1}{4}$ for the singlet channel and a factor of $\frac{3}{4}$ for the triplet channel.

The radial wavefunctions $g_{js}^{\ell\ell'}(r)$ are solutions of the Schrödinger equation, including the Coulomb and nuclear potentials (in this case, the REID93 soft-core potential [26]). The integration of the wavefunction is done with a simple leap-frog algorithm, starting at very low r ($\ll$ 1 fm) out to absurdly large r ($\approx$ 200 fm). At this large r, we read off the final normalization and phase of the radial wavefunctions and then renormalize the wavefunctions. We do this to repair any imperfect guessing of initial value of the wavefunction or its derivative near the origin. We use all ℓ's in the sum below $\ell_{max} = q_{max}r_{max}/\hbar c$.

6.3.2 genmaIMF1D, for Intermediate Mass Fragments

In the case of Intermediate Mass Fragment correlations, the kernel we use is[1]

$$K_0(q,r) = \theta(r - r_c)\,(1 - r_c/r)^{1/2} - 1\,. \tag{6.5}$$

Here, the reduced velocity is

$$v_{\mathrm{red}} = \frac{v}{(Z_1 + Z_2)^{1/2}}\,, \tag{6.6}$$

[1] Under the assumptions that the pair Coulomb correlation dominates the fragment correlation and that the fragments were approximately isospin symmetric.

and the distance of closest approach in (6.5) for symmetric fragments is approximately

$$r_c = \frac{2\,Z_1\,Z_2\,(A_1+A_2)\,e^2}{A_1\,A_2\,m_N\,v^2} \approx \frac{e^2}{m_N\,v_{\rm red}^2}\,. \qquad (6.7)$$

To get this kernel to function properly, it may be necessary to play with the units of q in `yourfile.controls`.

6.3.3 genma_meson, for meson pairs

All like-meson kernels can be generated by the `genma_meson` code, provided that we may neglect the nuclear force between the meson pair. For π and K mesons, this seems to be a reasonable approximation. Given this, the kernel for like mesons is

$$K_0(q,r) = \sum_{\ell=0,\ \ell\ \rm even}^{\infty} (2\ell+1)^{-1} |g_\ell(r)|^2 - 1\,. \qquad (6.8)$$

Here, $g_\ell(r)$ is the radial wavefunction that we obtain by solving the Klein-Gordon equation, including the Coulomb force. The Coulomb force is included using the minimal substitution prescription, so includes the relativistic correction to the Coulomb force.

The `genma_meson` code itself does not know about the mass or charge of the mesons in question. This is determined by the makefiles that build the `genmaPI1D` and `genmaKK1D` executables.

6.4 Using the kernels

Because it may be useful change the representation of the source in `image1D`, we decided that it is better if the `genma*1D` codes simply concentrate on building kernels, $K_0(q,r)$. In order to keep the details of the kernel separate from the imaging code, we have provided a function called `K_ftn(l,q,r)`[2] to serve as the generic program interface to the kernels. When called, `K_ftn(l,q,r)` first decides how to evaluate and return the value of $K_\ell(q,r)$. If there is no kernel on the disk, it uses its own copy of the non-interacting boson kernel (also provided by `genmaNI1D.f`). On the other hand, if there is a kernel on the disk it calls the routine `rawKspline`. Currently this function resides in the `rawKspline.f` file.

All of the kernel generators produce a table of kernel values on the same grid in r and q defined by Eq. (6.3) and specified in `gridinfo1D.inc`. To make use of this table, we have created a routine called `rawKspline` which creates a 6^{th} polynomial interpolated approximation to the kernel saved on disk. The `rawKspline` routine is also contained in `rawKspline.f`.

[2] For the one dimensional problem here, $\ell = l = 0$.

Chapter 7

Making Test Correlations

In this chapter, we discuss two methods for generating correlations for testing the imaging code and kernels. The first code, `makco1D`, convolutes one of a variety of sample source functions with a kernel. In other words, `makco1D` performs the integration in Eq. (1.11), which we repeat here:

$$\mathcal{R}(q) = 4\pi \int_0^\infty dr r^2 K_0(q,r) S(r). \qquad (7.1)$$

The other code is an extension of Scott Pratt's CRAB code [6] that we call `sourcemaker`. CRAB uses the single particle sources that comprise $S(r)$ to construct the correlation directly via:

$$\mathcal{R}(q) = \int d^4r \; K_0(q,r) \times \int d^3R \, D(R + r/2, \mathbf{P}/2) D(R - r/2, \mathbf{P}/2). \qquad (7.2)$$

To obtain the source, `sourcemaker` must then re-do the integrals in Eqs. (1.5) and (1.6), which we combine here:

$$S(\mathbf{r}') = \int dr_0' \, d^4R \, D(R + r/2, \mathbf{P}/2) D(R - r/2, \mathbf{P}/2) \qquad (7.3)$$

We will first explain how to invoke `makco1D` using the `createtest` script. Following this, we will explain some of how `makco1D` works, in the event that you wish to extend it. Finally, we give a brief overview of `sourcemaker` and the other codes in the `CRAB-addons` directory that are needed to use CRAB and HBTprogs together.

7.1 Running `makco1D`

As with the other codes, `makco1D` requires specific input filenames, so we have provided a script `createtest` in order to simplify the process of running `makco1D`. The input files and the required extensions are listed in Table 7.1.

To run `makco1D` using `createtest`, invoke it in the same manner you invoke `imageit`:

```
% ./createtest [prefix] [PID] [scratchdir]
```

The script `createtest` works in much the same way that `imageit` works. First, it copies all the relevant files to a scatch directory with names changed according to Table 7.2. It then invokes `makco1D`. Finally, the script copies all of the resulting files back to the starting directory with the new names as listed in Table 7.2.

Table 7.3 is a listing of all of the sources that `makco` can use to generate correlation functions. In all cases, the sources are all normalized to a parameter called λ.

7.2 How `makco1D` works

The program `makco1D` is very simple. It simply performs the convolution in Eq. (1.11) using a user specified source and the kernel chosen by `pick-kernel` and invoked through `K_ftn`. The `convolve` routine averages the correlation over the q_i bins in the input data file `yourfile.corin`. The `rintegral` routine performs the actual integral in Eq. (1.11). All integrals are done using the Romberg integration scheme (see [24]). For speed, the user specified source is placed in a table and a simple cubic spline interpolates this table of source values. After the correlation is generated, `makco1D` simulates statitical noise using the errors from the data in a correlation input file.

Input file extensions	
extension	file description
`.controls`	Main control file.
`.corin`	Correlation data.
`.dimensions`	FORTRAN include file that specifies various array sizes.
`.souin*`	Input source function for `makco1D` option 100.

Output file extensions	
extension	file description
`.cormod`	Generated test correlation, with errors and statistical noise.
`.cormod_noerr`	Generated test correlation, w/o errors and statistical noise.
`.souin`	The model source generated by `makco1D`.

*This file is overwritten on exit, but it is needed for option 100.

Table 7.1: List of the file extensions used by the `createtest` script.

Input Files	
starting name	working name
`yourfile.corin`	`corinput.dat`
`yourfile.dimensions`	`dimensions1D.inc`
`yourfile.controls`	`main.controls`
`yourfile.souin`	`souinput.dat`

Output Files	
working name	ending name
`souinput.dat`	`yourfile.souin`
`testcorr.dat`	`yourfile.cormod`
`noerrcorr.dat`	`yourfile.cormod_nerr`

Table 7.2: Mapping of starting filenames to working filenames and back to ending filenames in the `createtest` script.

7.3 Adding to `makco1D`

Adding sources to `makco1D` is straightforward, provided you watch for a few things:

1. The source must be normalized to 1 (see Eq. (1.7)).
2. The source has units of fm^{-3}.
3. Once you have included your source, add a menu entry in the `GetChoice` and `SimpleSource` routines in `makco1D`.

Alternatetively, you could be lazy and just use the "load source from disk" option from the `makco1D` menu.

7.4 CRAB and friends

As an alternative to `makco1D.f`, you might consider using Scott Pratt's CRAB code [6]. This code takes the freeze-out points from a transport model and computes the pair correlation function. The data itself must be formatted according to the OSCAR standard [29]. We have written `sourcemaker` to take the same OSCAR formatted set of freeze-out points and create a source function from them. This code uses the procedure outlined in Ref. [9].

In addition to `sourcemaker`, we provide a few additional files in the `CRAB-addons` directory. First, CRAB provides a code called `phasemaker` to generate the freeze-out positions from a thermal Gaussian single particle source, $D(r, \mathbf{p})$. We have extended this code to generate single particle sources with exponential halos. Second, in order to help CRAB and HBTprogs coexist, we have provided a preconfigured driver and binning. Finally, CRAB's output format differs from HBTprogs's `yourfile.corin` file format so we wrote a converter code called `CRAB-converter.cpp`.

Menu Item	Equation of Source Function	Notes
Gaussian	$\frac{\lambda}{(2\sqrt{\pi}R_0)^3}\exp(-r^2/4R_0^2)$	
Gaussian with Tail	$\lambda\left(\frac{1}{2}\frac{1}{(2\sqrt{\pi}R_0)^3}\exp(-r^2/4R_0^2)+\frac{1}{2}\frac{15}{4\pi^5R_1^4}\frac{r}{\exp(r/R_1)-1}\right)$	
Sharp Sphere	$\lambda\frac{3}{4\pi R_0^3}\theta(R_0-r)$	
Sqrt Thingee	$\lambda\mathcal{C}\theta(R_0-r)\left(1-\exp\left(\sqrt{1-(r/R_0)^2}\right)\right)$	$\mathcal{C}$ is a normalization constant determined by the program
Gaussian w/ cutoff	$\lambda\theta(R_{cut}-r)\frac{\mathcal{C}}{(2\pi R_0^2)^{3/2}}\exp(-r^2/4R_0^2)$	$\mathcal{C}$ is a normalization constant determined by the program
Hard Sphere Freeze-Out Density	$\lambda\frac{9}{\pi R_0^3}\theta(R_0-r)\left[\frac{2}{3}-\frac{r}{R_0}+\frac{1}{3}\left(\frac{r}{R_0}\right)^3\right]$	This corresponds to a freeze-out phase-space density $f(r,p)\propto\theta(r-R_0/2)$
Shell Freeze-Out Density	$\lambda\frac{1}{2\pi R_0^3}\theta(R_0-r)\frac{R_0}{r}$	This corresponds to a freeze-out phase-space density $f(r,p)\propto\delta(r-R_0/2)$
Exponential	$\lambda\frac{1}{8\pi R_0^3}\exp(-r/R_0)$	
Woods-Saxon	$\lambda\frac{\mathcal{C}}{1-\exp((r-R_0)/R_1)}$	$\mathcal{C}$ is a normalization constant determined by the program
Delta Function	$\lambda\frac{1}{4\pi R_0^2}\delta(r-R_0)$	
Dipole	$\lambda\frac{2}{\pi^2}\frac{R_0}{(r^2+4R_0^2)^2}$	
Sum of Any Two Shapes	n/a	The user chooses which shapes.
Load from disk	n/a	Needs `souinput.dat` file and code overwrites it on exit.

Table 7.3: List of sources used by `makco`.

Chapter 8

Frequently Asked Questions (FAQ)

1. **What happens if my correlation doesn't go to 1 at large q?** You most likely will end up with a source that is meaningless, namely it oscillates wildly and is mostly negative (that is, if it didn't crash `image1D` already). Unfortunately, there are many reasons why your correlation might not get to 1 at large q. The trivial one, namely that the correlation is normalized wrong, should be fixed *before* attempting to image. However, various kinds of flow-induced position-momentum correlations are known to result in correlations that do not go to 1 [27]. This type of problem seems to be very common in low energy proton-proton correlations.

2. **My restored correlation or my model correlation oscillates!** This is most likely either due to a bug/misuse of the codes, but may also be due to there being an edge in the source. See [16] for some detail on this.

3. **I'm using `BSplineDegree=0` and my 2nd bin is really low!** This can usually be fixed by increasing `BSplineDegree`. This has the effect of tying the first few bins together in a bigger spline. You won't loose resolution, but you will squash many unphysical oscillations. As a side benefit to going to higher degree Basis splines, you will be able to use various equality constraints to stablise the image at low r.

4. **My source tail oscillates wildly!** You may either have some tweaking to do or you don't have the resolution to resolve the source at large r. The constraints that control the large r behavior (`zero_source_at_large_r` and `smooth_source_at_large_r`) may help or you may be forced to decrease the size of your imaging region.

5. **My source has a cusp at large r!** This is most likely an artifact of aliasing caused by the statistical noise in the correlation. You can either choose to ignore it or squash it with the two constraints that control the large r behavior of the source (`zero_source_at_large_r` and `smooth_source_at_large_r`).

6. **My source has a cusp at low r!** You can fix this easily by adding the `diff_constraint` constraint. Of course, one must be carefull because you may indeed have sharp structure in your source and may not be able to resolve it once the constraint is turned on.

7. **My head has a cusp!** Sorry, I can't help you there.

8. **Can I have variable sized bins in my data?** Yes, reread section 4.2.

9. **How can I get the integral of the source?** It is already automatically computed by `image1D`. You can control the upper limit of integration with the option `up_int_lim` in your main controls file.

10. **What are some of the issues with CERNLIB?** When the project was started, using CERNLIB made a lot of sense: it was free, easy to use, and had most of the functions we needed for the code. In particular, it has the `NORBAS` Basis spline package and `minuit`. Unfortunately, over time, CERNLIB went from useful package to crutch:

(a) `minuit` is flaky on linux. Even on the Compaq alpha's `minuit` has been known to freeze when searching for optimal knots. Incidentally, when this happens, changing `up_fit_lim` a little bit usually fixes the problem.

(b) overuse and misuse of common block cause trouble when parallelizing the code

(c) archaic and often obsolete FORTRAN

(d) CERN is no longer supporting it

We are working to remove all need to link to this code by replicating CERNLIB functionality in the `libplay.a` library in the `lib/` directory.

11. **How can I pick good knots?** Read the section 5.1, or better yet, read Refs. [3, 21].

12. **Why don't my Gaussian fit and my source look the same?** There are many reasons, but the obvious one is that your source isn't Gaussian! Of course, if you are using `makco1D` and you *put in* a Gaussian, then most likely there is either an error in your input files or you have some tweaking to do. It is also possible that, given the binning of you data, that you really can't resolve part (or all) of your source. If this is the case, consider refining the binning of the correlation model you have. On the other hand, if you are modeling real data and you really can't change the binning of the correlation, it may be that a Gaussian fit is all you can really do.

13. **Will there be other representations of the source in the code in the future?** Probably not. We are considering a full rewrite in `c++` which would allow for this, but we are not planning to add this feature to this FORTRAN code.

14. **How can I make a kernel when I don't know the potential?** Most likely, you are in trouble. Sometimes though, if you know the scattering length and or a few phaseshifts, you can work out the wavefunction and/or potential. See refs. [18, 26] for a starting point.

15. **Why don't you use inequality constraints in the inversion?** We'd like to. An old version of this code actually did support it, but the error estimation and propagation in the code was suspect so this feature was removed. To put it back in, we need to do a lot of reading about "Active set methods." If you really want it, you could figure it out and we'll be happy to put it in...

Chapter 9

CONCLUSIONS

If you have comments, suggestions for improvements, or bug fixes (or even better, software patches!) do not hesitate to contact one of the authors. In fact, we encourage it. Currently there are several areas of the code that could use substantial enhancement:

1. A bigger selection of kernels
2. Simpler user interface (porting FORTRAN 90 would allow use to dynamic memory and hence remove the include files specifying the array dimensions)
3. Use of inequality constraints
4. Use of the full data covariance matrix

The TODO list in Appendix C has a more complete list.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48. This work was also performed under the National Science Foundation Grant NSF-....

Appendix A

File Manifest

A.1 Main Files

1. Directories:

   `config/`: configuration files

   `doc/`: the documentation

   `doc/example/`: a simple pp example

   `doc/plot_templates/`: `xmgrace` plot template

   `include/`: FORTRAN include files

   `kernels/`: any pre-built kernels

   `lib/`: files to make the shared library

   `sbin/`: the scripts get installed here

   `src/1d/`: main source codes

   `src/kernel_generators/`: kernel generator source codes

   `src/scripts/`: unprocessed scripts

   `src/CRAB-addons/`: extra codes for use with `CRAB`

2. Documentation:

   `CHANGELOG`: log of major changes with each version

   `README`: read this first

   `RELEASE`: release version

   `TODO`: to do list

   `COPYING`: notes on copying and distributing the codes

   `NEWS`: no news is good news...

   `AUTHORS`: list of the authors

   `INSTALL`: `autoconf`-generated installation instructions

   `HBTmanual.tex`: this manual.

   `HBTmanual.ps`: this manual in postscript format.

   `emission.eps`: a figure in the manual.

   `bsplines.eps`: another figure in the manual.

   `example_plots.eps`: yet another figure in the manual.

3. Kernel generators:

   `genmaIMF1D.f`: IMF kernel generator

   `genmaNI1D.f`: kernel generator for non-interacting spin-0 bosons

   `genmaPP1D.f`: proton kernel generator

   `genma_meson.f`: combined 1D and 3D meson kernel generator

4. Other Programs:

   `image1D.f`: the imaging code

   `makco1D.f`: code to generate test correlations

   `fitsource1D.f`: code to fit Gaussians to imaged sources

   `optres1D.f`: undocumented feature

   `share1D.f`: code shared by several of the 1D codes

   `viewsource1D.f`: for viewing the source, without inverting again

   `mtx2xpm.cpp`: gizmo to make a grey-scale pixmap of a matrix

5. Scripts:

   `configure`: main configuration script generated by `autoconf`

`config.status`: configuration script used by `configure`

`createtest`: script for using `makco1D`

`pick-kernel`: script for building and using the kernels

`imageit`: script for using `image1D`

6. Misc. Files:

`Makefile.in`: template `Makefile` (used by `configure`)

`configure.in`: template `configure` file for use with `autoconf`

`configure`: script to generate `Makefiles` for different machines (generated by `autoconf`

7. Include Files:

`imfs.inc`: particle properties for the IMF kernel

`kaons.inc`: particle properties for the kaon kernel

`pions.inc`: particle properties for the pion kernel

`protons.inc`: particle properties for the proton kernel

`constants.inc`: constants of the universe. Unless you have warp capabilities, don't mess with these.

`gridinfo1D.inc`: size of grid used in 1D kernels

A.2 The example files

1. Control/Include Files:

`example_pp.controls`: main control file

`example_pp.bspline`: include file that sets the b-spline order

`example_pp.dimensions`: include file that sets the dimensions of the image model space and the data space

2. Input Data:

`example_.souin`: sample input source. It is a Gaussian with $R_0 = 4$ fm.

`example_pp.corin`: sample correlation using `example_.souin`.

3. Outputted Source:

`example_pp.imag`: sample restored source from `example_pp.corin`.

`example_pp.knots`: knots that define the b-spline.

`example_pp.coeffs`: coefficients of the b-spline expansion.

`example_pp.covmtx`: covarience matrix of the coefficients

`example_pp.restored`: restored correlation (compare to `example_pp.corin`).

`example_pp.bestfit`: results of a Gaussian fit to the source image.

`example_pp.optres`: output of an obscure, undocumented feature.

4. Logs:

`example_pp.runlog`: output of `imageit`.

`example_pp.knotlog`: log of `minuit`'s knot-finding adventures.

`example_pp.fitlog`: output of `fitit`.

A.3 CRAB addons

1. Documentation:

`README_CRAB-addons`: what passes for documentation for this set of codes

2. Main Files:

`crab_HBTprogs.cpp`: drop in replacement for `crab.C`

`crab_bindefs_HBTprogs.cpp`: drop in replacement for a `CRAB` binning file

3. Utility Codes:

`crab-converter.cpp`: converts `CRAB` output to a `corinput.dat`-like formatted correlation

4. Phasemaker:

`phasemaker.cpp`: modified version of Scott Pratt's code – this version generates Gaussian sources with exponential halos

5. Sourcemaker:

`sourcemaker.cpp`: code to generate source functions from an OSCAR formatted particle freeze-out distribution

`randfuncs.hpp`: code to generate random numbers with various distributions

`threevectors.hpp`: simple three-vector class

`fourvectors.hpp`: simple four-vector class

`fv_tester.cpp`: tester code for the three and four vector classes

Appendix B

CHANGELOG

```
CHANGELOG file
last updated 18 Jan 2002
+-----------+
|version 1.0|
+-----------+------------------------------------------------------------
Overall:
--------
1) new autoconf/automake configuration and build scheme
2) new control scripts (hopefully more rational and more portable)
3) new directory structure
4) removed 3d codes.  save those for version 2.0
5) finished documentation

makeco1D.f:
-----------
1) fixed option 100 (load from disk)
2) fixed case when data has un-equally spaced bins

genmaPP1D.f:
------------
fixed Coulomb bug introduced sometime in last few months

+-----------+
|version 0.9|
+-----------+------------------------------------------------------------
Overall:
--------
1) needlessly inflated the version number
2) rewrote setup script in python

rawKspline.f:
-------------
1) unified all kernel loaders in one routine
2) unified all kernel functions in one routine
3) switched to higher order polynomial for interpolating kernel table
4) added meta-data to kernel files
```

```
+--------------+
|version 0.09.3|
+--------------+-------------------------------------------------------
In image1D:
-----------
errors were computed incorrectly -- it is now fixed and the errors on
S(r), integral of S(r), and restored correlation are much smaller now.

In lib/:
--------
random number generator changed so that it generates a different sequence
of random numbers each time it is called

+--------------+
|version 0.09.2|
+--------------+-------------------------------------------------------
In the genmaXX files:
---------------------
1) renamed them from genmaXX -> genmaXX1D because there are also
genmaXX3D kernels in the works.  The genma_meson.f code seems
to be an exception to the naming scheme.  Well, it is capable
of producing 1D *and* 3D kernels for any mesons (you have to
change particle.inc however)
2) the format for saving the kernel is different.  now the format is
formatted and it specifically is:
   NSQ NSR
   K_0(1,1) K_0(1,2) K_0(1,3)....
   K_0(2,1) ...
   .
   .
   .
   l=0
   K_1(1,1) K_1(1,2) ....
   .
   .
   .
   l=1
   ...
here, NSR and NSQ are the number of steps in the r and q directions.
K_l is the lth componant of the kernel (expanded in legendre polynomials
for 3D work).
3) there are alot more bins in r and q.  also, the way i have it rigged
now, you don't have to rebuild the kernel each time you change the binning
in q
4) no longer used interpolation to speed up the potentials.  since the
kernel doesn't have to be remade so often, it can take longer to build

In makco:
---------
1) makco has more choices
2) more accurate because kernel better
```

```
In the image1D:
---------------
1) b-splines
2) equality constraints
3) checks inequality constraints
4) optimized knots found using faster/better minimization routine
5) can output the optimal resolution matrix (some day i'll have to explain
what this is)

Other stuff:
------------
1) is a 3D version of image1D (it was in the first version of the new code
i put on the web, but i took it out of the last one as it's not really
done testing yet)
2) viewsourceXD codes, so you can remake the images if you only have the
coefficients and knots.
3) mtx2xpm, a little c++ code to make any of the matrices made by the
codes into a grey-scale pixmaps
4) fitsource1D, a little code that fits gaussians to imaged sources and a
script fitit to automate the process
5) genmaDA1D and smearker (thanks Giuseppe!)

Overall:
--------
1) centralized controls whereever possible (e.g. main.controls)
2) fewer include files to change and they are named more intelligently.
3) using CERNLIB whereever possible (it's reasonably fast and available
everywhere)
4) runs on more platforms:  wrote it on Linux (egs f77) and on OSF (dec
fortran)
5) scripts for automating some boring things and for giving more
reasonable filenames to things


+------------+
|version 0.03|
+------------+-------------------------------------------------------
1) added option of inputing source from disk into makco.f
   (this allows you to build model sources using other programs like RQMD)
2) simplified Makefile
3) fixed linking bug in installib.com (thanks to Dennis Reichhold 4/7/99)

+------------+
|version 0.02|
+------------+-------------------------------------------------------
1) minor bug on some systems in lib/simplx.f
2) minor fixes in makefile that allow programs to run under Linux
3) added Woods-Saxon like source to makco.f
4) added realistic noise to sources from makco.f
5) fixed bug that made standard deviation of noise in randomnoise sqrt(2)
```


too big
version 0.01
original version

Appendix C

To Do List

1. Merge 3D codes
2. Merge Giuseppe's deutron-alpha kernel
3. Merge Giuseppe's `smearker`
4. Clean up interface
5. Option to fix first knots (as in old code), option to let *all* knots float
6. Get opt. knots in 3D code (possibly use 1D code for this)
7. Tweak opt. knot routines in 1D, also maybe abstract them so can use in 3D
8. More metadata in `kernel.dat` and `kmtx.dat` files
9. Fix norm constraints
10. Merge `knotlist` and `source_coeff`
11. Generic kernel generator
12. Frame dependence of 3D analysis
13. Quality checks of outputted covariance matrix (is it symmetric and positive definite?)
14. Remove dependence on CERNLIB

Bibliography

[1] David A. Brown and Paweł Danielewicz. "Imaging of sources in heavy-ion reactions." *Phys. Lett. B* 398: 252–258 (1997).

[2] David A. Brown and Paweł Danielewicz. "Optimized discretization of sources imaged in heavy-ion reactions." *Phys. Rev. C* 57: 2474–2483 (1998).

[3] David A. Brown and Paweł Danielewicz. "Observing non-Gaussian sources in heavy-ion reactions." *Phys. Rev. C* 64: 014902 (2000).

[4] S. E. Koonin. "Proton pictures of high-energy nuclear collisions." *Phys. Lett. B* 70: 43–47 (1977).

[5] S. Pratt, T. Csörgő, and J. Zimányi. "Detailed predictions for two-pion correlations in ultrarelativistic heavy-ion collisions." *Phys. Rev. C* 42: 2646–2652 (1990).

[6] CRAB source code. `http://www.nscl.msu.edu/~pratt/freecodes/crab/`.

[7] H. Barz. "Effects of nuclear coulomb field on two meson correlations." *Phys Rev. C* 53: 2536 (1996).

[8] L. Martin, C.K. Gelbke, D. Erazmus, and R. Lednicky. "Influence of the emitter coulomb field on two proton correlation function." *Nucl. Phys. A* 604: 69 (1996).

[9] S. Y. Panitkin and D. A. Brown. "Imaging proton sources with space-momentum correlations." *Phys. Rev. C* 61: 021901 (2000).

[10] S. Pratt. "Pion lasers from high-energy collisions." *Phys. Lett. B* 301: 159 (1993).

[11] S. Pratt. "Deciphering the centauro puzzle." *Phys. Rev. C* 50:469 (1994).

[12] S. Pratt. *Physics of the Quark-gluon Plasma*, ed. R. Hwa, World Scientific, Singapore (1996).

[13] Q.Q. Chao, C.S. Gao, Q.H. Zhang. *J. Phys. G* 21: 847 (1995).

[14] Paweł Danielewicz and Peter Schuck. "Formulation of particle correlation and cluster production in heavy–ion–induced reactions." *Phys. Lett. B*, 274:268–274 (1992).

[15] S. Pratt. "Validity of the smoothness assumption for calculating two-boson correlations in high-energy collisions." Phys Rev. C 56: 1095-1098 (1997).

[16] D. A. Brown, F. Wang and P. Danielewicz. "Implications of the unusual structure in the pp correlation from Pb+Pb collisions at 158 AGeV." *Phys. Lett. B* 470, 33 (1999).

[17] S. Pratt. "Coherence and Coulomb Effects on Pion Interferometry." *Phys. Rev. D* 33: 72-79 (1986).

[18] D. Boal, C.K. Gelbke, and B. Jennings. "Intensity interferometry in subatomic physics." *Rev. Mod. Phys.* 62: 553-602 (1990).

[19] U. Heinz and B. Jacak. "Two particle correlations in relativistic heavy–ion collisions." *Ann. Rev. Nucl. Part. Sci.* 49: 529-579 (1999).

[20] U.A. Wiedemann and U. Heinz. "Particle interferometry for relativistic heavy–ion collisions." *Phys. Rep.* 319: 145-230 (1999).

[21] C. de Boor. *A Practical Guide to Splines*, Springer-Verlag, (1978); MRC 2952 (1986) in *Fundemental Developments of Computer-Aided Geometric Modeling*, L. Piegl (ed.), Academic Press, (1993).

[22] CERN Program Library. `http://wwwinfo.cern.ch/asd/cernlib/`.

[23] A. N. Tikhonov. "Solution of incorrectly formulated problems and the regularization method." *Sov. Math. Dokl.* 4:1035–1038 (1963).

[24] W.H. Press et al. *Numerical Recipes: The Art of Scientific Computation.* Cambridge University Press, Cambridge (1992).

[25] A.Tarantola. *Inverse Problem Theory.* Elsevier, Amsterdam (1987).

[26] V. G. J. Stoks et al. "Construction of high quality nucleon nucleon potential models." *Phys. Rev. C* 49: 2950–2962 (1994).

[27] T. Trainor and J.G. Reid. "Space-time autocorrelations and Hubble flow: probing small length scales in heavy ion collisions." e-Print Archive: hep-ph/0004258.

[28] A. Messiah. *Quantum Mechanics*, Dover Pub., Mineola, NY USA (1999).

[29] OSCAR Standard Codes and Routines. `http://www-cunuke.phys.columbia.edu/OSCAR/`

```
+-----------------------------------+
|          HBTprogs                 |
|         version 1.0               |
|                                   |
| authors: David A. Brown           |
|          Pawel Danielewicz        |
|                                   |
+-----------------------------------+
```

This is the README file for a set of programs that we developed to invert angle averaged correlation functions. You are free to use and modify the codes, we only ask that you reference either the manual or references [1,2,3] in any publications using results obtained with these codes. We are not responsible for any misuse of these codes. If you have any questions, comments, suggestions or bug fixes regarding the programs or the documentation, please contact David Brown at brown170@llnl.gov or Pawel Danielewicz at danielewicz@nscl.msu.edu.

QUICKSTART

1) unpack the tarball:
   % gunzip HBTprogs.tar.gz
   % tar xf HBTprogs.tar
2) run configure:
   % ./configure
2) do a make install-data:
   % make install-data
3) read the manual in doc/
4) rerun the example in doc/example/

IF YOU HAVE PROBLEMS

If you have problems, there are two immediate sources of information:

1) The INSTALL file which discribes the various options of the configure script
2) The manual doc/HBTmanual.ps

If this doesn't help, contact David Brown. His contact information is listed in the AUTHORS file.

REFERENCES

[1] David A. Brown and Pawel Danielewicz.
Imaging of sources in heavy-ion reactions.
Phys. Lett. B, 398:252--258, 1997.

[2] David A. Brown and Pawel Danielewicz.
Optimized discretization of sources imaged
in heavy-ion reactions.
Phys. Rev. C, 57(5):2474--2483, May 1998.

[3] David A. Brown and Pawel Danielewicz.
Observing non-Gaussian sources in heavy-ion reactions.
Phys. Rev. C 64: 014902 (2000).