Spectral Reconstruction via Backus-Gilbert

Handover Notes

Benjamin Page
benjaminpage.acer@gmail.com
b.t.page@swansea.ac.uk

April 20, 2024

Contents

Preface

These notes constitute a brief overview of the Backus-Gilbert inversion code and should contain all the necessary information to compile generation scripts and obtain the Backus-Gilbert coefficients. Any questions may be freely directed to either my primary email (benjaminpage.acer@gmail.com) or my institutional email (currently b.t.page@swansea.ac.uk).

This work was performed in conjunction with Chris Allton and Antonio Smecca.

1 Installation

Installation instructions along with the necessary dependencies can be found in `https://gitlab.com/fastsum/vortex`. The main non-standard dependencies are:

- GMP (`https://gmplib.org/`)
- MPFR (`https://www.mpfr.org/`)
- GVAR (`https://github.com/gplepage/gvar`)
- ZKCM (`https://sourceforge.net/projects/zkcm/`)

Each of these will (probably) have to be installed from source (with the exception of GVAR, which is available via `pip`), with installation instructions available on each web page. **OpenMP is also required for compilation– this should come as standard with `gcc` and `g++` but may need to be installed.** Note: Only branch `backus` is runs in parallel– branch `hmr` is coded for parallel running but still runs in serial[1]

The following dependencies **must be installed in the following order:**

1. GMP
2. MPFR
3. ZKCM

Note: on MacOS, GMP and MPFR may be installed using Homebrew via `brew install`.

Within each branch there is a Makefile which compiles the code and libraries. This is achieved simply by using the command `make` in the directory created by cloning the repository.

1.1 Setting the linker library path

If executing the code raises the error

```
./backus: error while loading shared libraries:
libinterface.so: cannot open shared object file: No such file or directory
```

then the linker library path has not been correctly set. This can be checked using

```
echo $LD_LIBRARY_PATH
```

which should give the expected location(s) of the libraries, which should contain the interface and ZKCM call libraries, `libinterface.so` and `libzcall.so`. Usually these are stored in `lib/` if building from the directory created by cloning the repository.
The library path can be set adding

[1] I suspect this is because the code is parallelised over the constraint vector loop (see Appendix C) which I think executes faster than the thread sertup and allocation time.

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:lib/"
```

to the bash profile (`./bashrc` or equivalent). Executing this command in the shell **will only set the linker library path for that current shell session**.

2 Gen-2L Configurations [IMPORTANT]

2.1 File format

The Gen-2L configurations which the I/O utility code is designed to read is not in the format available on the filestore[2], but instead the sequential "`*onia_Nt_Nc`" format created by Chris Allton's Fortran converter code.

2.2 Binary naming convention (redundant for `backus` and `hmr` branches

The I/O code has a translation layer which reads the relevant channel from the configuration files. This is achieved by indexing the channels using a binary naming scheme rather than the channel names set out by Don Sinclair. The utility code has functions which convert between the binary name and Don Sinclair's naming convention.

D. Sinclair's Scheme	Binary Scheme
spp_0	s00
spp_i	s01
sxx_0	s10
sxx_i	s11
⋮	⋮

The binary naming scheme is now essentially redundant and was designed for streamlined reading of the s-wave channels. This naming scheme has been dropped for the `backus` and `hmr` branches, but still may be referenced in certain places and comments.

3 Branches

The code is separated into three branches: `backus`, `hlt` and `hmr`. The `backus` branch contains the original area-constrained second-moment approach proposed by Backus and Gilbert, along with an unconstrained, least-squares approach also proposed by Backus and Gilbert, and further developed upon by D. Oldenburg. The `hlt` branch contains adapted, unconstrained, least-squares code which can accept user-defined target functions and also outputs the value of the inverted kernel width matrix for later analysis. Note that the `hlt` branch is stale. Finally, the `hmr` branch (so named for the Hansen-Meyer-Robaina approach to the Backus-Gilbert method) contains area-constrained, least-squares code for coefficient generation.

4 Branch Contents

Below is a summary of the key files in each of the branches

4.1 Interface files

The matrix inversion(s) are performed via singular value decomposition using the C++ ZKCM[3] library. In order to facilitate the use of C++ code by the underlying C main script, a translation/interface layer is required. This functionality is performed by the following files and headers:

- `interface.cpp`

[2] I obtained the configurations from Tom Spriggs in 2020.
[3] `https://sourceforge.net/p/zkcm/home/Home/`

- `interface.h`
- `zcall.cpp`
- `zcall.h`

These interface files are compiled into library objects by the makefile and are needed by the coefficient generator code (hence the linker library path).

5 The Backus-Gilbert Method

The Backus-Gilbert method may be used to estimate solutions to a problem of the form

$$G(\tau) = \int K(\tau, \omega)\rho(\omega)d\omega \tag{1}$$

where $G(\tau)$ are a set of known, discrete observations and $K(\tau, \omega)$ is a known kernel function which maps the underlying, unknown function $\rho(\omega)$ onto $G(\tau)$.
The Backus-Gilbert estimate of $\rho(\omega)$ about some point $\omega_0 \in \{\omega\}$ is given by

$$\hat{\rho}(\omega_0) = \int A(\omega, \omega_0)\rho(\omega)d\omega \tag{2}$$

where $A(\omega, \omega_0)$ is called the sampling function or smearing kernel which produces the estimate from the spectrum and is constructed from a linear combination of the kernel functions $K(\tau, \omega)$ via:

$$A(\omega, \omega_0) = \sum_\tau c_\tau(\omega_0)K(\tau, \omega) \tag{3}$$

where $c_\tau(\omega_0)$ are a set of coefficients to be determined by the method. This is achieved by minimising some constraint on the shape of $A(\omega, \omega_0)$, parameterized by the functional $W[c_\tau]$ (see §6 for examples), such that the optimal estimate is found when $\partial_{c_\tau} W[c_\tau] = 0$.
The Backus-Gilbert estimate of the reconstructed spectrum is then given simply by

$$\hat{\rho}(\omega_0) = \sum_\tau c_\tau(\omega_0)G(\tau) \tag{4}$$

In practise, due to the inverse problem the solution to $\partial_{c_\tau} W[c_\tau] = 0$ requires the inversion of one or more near-singular matrices, $\mathcal{K}$. These are generally treated via the addition of a whitening or regularising matrix which introduces a trade-off between systematic and statistical error that must be removed using a subsequent optimizations scheme. Two common options in the literature are:

Tikhonov Regularisation:

$$\mathcal{K}(\alpha) = A(\alpha)\mathcal{K} + B(\alpha)\mathbb{1} \tag{5}$$

Covariance Regularisation:

$$\mathcal{K}(\alpha) = A(\alpha)\mathcal{K} + B(\alpha)\frac{\mathrm{Cov}[G(\tau)]}{G(0)^2} \tag{6}$$

where $A(\alpha)$ and $B(\alpha)$ are functions of the regularisation strength parameter α and control the shape of the trade-off curve. Two possible options are a simple linear addition,

$$A(\alpha) = \mathbb{1}, \qquad B(\alpha) = \alpha, \tag{7}$$

or a convex trade-off,

$$A(\alpha) = (1 - \alpha), \qquad B(\alpha) = \alpha. \tag{8}$$

6 Generating Coefficients

The type of coefficients which are generated depend on the branch type and main file used, with each of the available criteria listed below:

Branch: `backus`

source: `bgv6_spread.c`

$$W_1[c_\tau] = 12 \int_{\omega_{\min}}^{\omega_{\max}} (\omega - \omega_0)^2 A(\omega, \omega_0)^2 d\omega - \lambda \left[\int_{\omega_{\min}}^{\omega_{\max}} A(\omega, \omega_0) d\omega - 1 \right] \tag{9}$$

where λ is a Lagrange multiplier which is calculated by the source file (but not saved or stored). See Appendix A for a derivation of the minimised coefficients.

source: `bgv6_leastsq.c`

$$W_2[c_\tau] = \int_{\omega_{\min}}^{\omega_{\max}} \left[A(\omega, \omega_0) - \delta(\omega - \omega_0) \right]^2 d\omega. \tag{10}$$

See Appendix B for a derivation of the minimised coefficients.

Branch: `hlt`

source: `bg_hlt.c`

$$W_3[c_\tau] = \int_{\omega_{\min}}^{\omega_{\max}} \left[A(\omega, \omega_0) - \Delta(\omega, \omega_0) \right]^2 d\omega. \tag{11}$$

where $\Delta(\omega, \omega_0)$ is the target kernel (for example, one may choose the gaussian function $\mathcal{N}(\mu, \sigma)$ width with σ located about $\mu = \omega_0$ as the target). Note: the above criterion ideally requires the minimisation constraint on the averaging functions to preserve the scale of the spectrum. This was not implemented as I had not yet solved the minimisation problem when this code was created in 2022/2023.

Branch: `hmr`

source: `bgv6_hmr.c`

$$W_3[c_\tau] = \int_{\omega_{\min}}^{\omega_{\max}} \left[A(\omega, \omega_0) - \delta(\omega - \omega_0) \right]^2 d\omega - \lambda \left[\int_{\omega_{\min}}^{\omega_{\max}} A(\omega, \omega_0) d\omega - 1 \right] \tag{12}$$

See Appendix C for a derivation of the minimised coefficients.

6.1 Generating the correlator input file

The source file requires the following information line-by-line:

Line Number	Description
1	N_τ, the temporal extent of the lattice.
2	N_s, the number of slices to sample (corresponding to $N_s + 1$ sample points ω_0)
3	$G(\tau)$, the **full** Euclidean correlator.
$\vdots$	Note: when choosing a Euclidean time window $\tau \in [\tau_1, \tau_2)$ via the command line
$N_\tau + 2$	arguments, the source file automatically truncates the correlator accordingly.
$N_\tau + 3$	
$\vdots$	$\mathrm{Var}[G(\tau)]$, the variance.
$2N_\tau + 2$	
$2N_\tau + 3$	
$\vdots$	Upper triangle of the covariance matrix, $\mathrm{Cov}[G(\tau), G(\tau')]$ for $\tau \neq \tau'$.
EOF $\left[\frac{N_\tau(N_\tau) - 1}{2} \right]$	

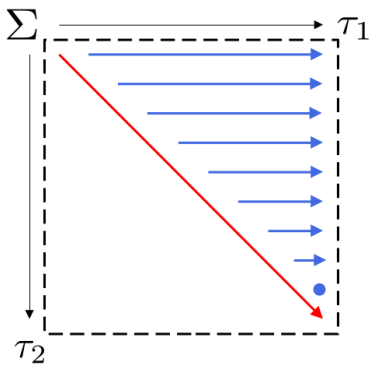

Figure 1: Pattern with which the `gfetch.sh` code traces out the covariance matrix for the output file. First the main diagonal (red, variance) is printed, then the off-diagonal elements (blue) are printed for each τ_2 in increasing steps of τ_1. Since the covariance matrix is symmetric, we only need one half of the matrix to reconstruct the full covariance in the coefficient code.

This information is readily created using the `gfetch.sh` bash script, provided it has been pointed to the Gen-2L directory. See Fig. 1 for how the covariance matrix should be formatted in the output file created by `gfetch.sh`. The output file is prepared by executing the following command:

```
./gfetch.sh <channel> <Nt> <Ns> <Nb>
```

where `channel` is the channel in Don's naming scheme (e.g. 'spp_i'), `Nt` is the temperature/temporal extent, `Ns` is the number of sampling slices for the reconstruction (i.e. create `Ns`+1 coefficients) and `Nb` is the number of bootstrap samples to draw from. Passing `Nb=-1` will use the whole set of configurations to calculate the covariance matrix.
It is recommended to prepare the input information as a separate text file using `gfetch.sh` so that it can be piped into the source file, i.e.

```
$ cat output.dat | ./backus_file <args>
```

6.1.1 `covgen` vs `covgend`

The covariance matrix calculation is handled by the `covgen` and `covgend` executables. `covgen` calculates the covariance matrix assuming the whole set of configurations is used (this is the "static" covariance approach and is **used by default in `gfetch.sh`**). `covgend` calculates the dynamical covariance matrix (i.e. dependent on the bootstrap samples drawn) and can be selected by specifying the type using the `gentype` variable in `gfetch.sh`.

6.2 Command-line Arguments

The Backus-Gilbert coefficients are generated by supplying the relevant source file (see above) with the following command line arguments:

g – Operating mode [0, 1 or 2]:

- 0 - Print the spectral estimate only. No saving to file.
- 1 - Save the coefficients to file.
- 2 - Save the coefficients and the kernel weight matrix to file (only for files with a single matrix inversion).

$\omega_{\min}$ – Minimum of the sampling range.

$\omega_{\max}$ – Maximum of the sampling range.

α – The regularisation parameter.

τ_1 – The initial Euclidean time.

τ_2 – The final Euclidean time.

Note that 'g' must be greater than 0 else the code does not save the coefficients.

6.3 Regularisation

The exact regularisation method can only be changed by altering the source file. Each source file allows for swapping between Tikhonov and covariance regularisation using the `tikh` variable, where `tikh=1` denotes Tikhonov whitening and `tikh=0` denotes covariance whitening. If covariance whitening is used, the regularisation can be restricted to the main diagonal only (i.e. variance-only whitening) by changing `emode` from 1 (full whitening) to 0.

Note that there is no readily available method for altering the form of the trade-off shape functions $A(\alpha)$ and $B(\alpha)$, so these will need to be changed as-required. Currently the code uses the shape functions defined in Eq. (7). Changing these shape functions will require editing the code, with which care must be taken due to the MPFR arithmetic (see §6.4).
Note: if using the `hmr` branch and using a shape function such that $A(\alpha) \neq \mathbb{1}$ then the sampling vector C_τ (see Appendix C) should[4] also be multiplied by $A(\alpha)$. In the coefficient code, the sampling vector is called `KSamp[]`

6.4 Setting arbitrary precision

The coefficient generation code is set up to support arbitrary precision. The matrix inverses are achieved by side-stepping into C++ and performing the singular value decompositions using ZKCM, as the LAPACK and LINPACK C libraries do not support quadruple precision or better. The standard precision is 128 bits (quadruple precision), but the coefficient generation code can support arbitrary precision, set using the `prec` variable in the coefficient code. Note that increasing the precision will naturally increase both compute time and memory requirements.
Note that all arithmetic operations use MPFR functions and not the standard C operators– this is a facet of the multiprecision library. A list of the MPFR arithmetic functions can be found at `https://www.mpfr.org/mpfr-current/mpfr.html#Arithmetic-Functions`.

7 Analysis scripts

Once the coefficients have been generated, reconstructing the spectrum from the Gen-2L correlators is straightforward.
The analysis repository found at `https://github.com/jadot-bp/backus-analysis` contains some utility code and an analysis script containing some minimum-working examples for fetching correlator data, reconstructing the spectrum, generating averaging functions etc.

7.1 `backus_utils.py`

This is the utility script which offloads some of the more complex I/O functions such as reading the correlator files, bootstrapping, covariance matrix estimation and more.

7.1.1 Modifying correlator I/O

If you wish to use correlators in a different format from those described in [link] then you **must** modify the following functions:

- `get_valid_selection` – This function returns a list of the available correlator configuration filenames from which the samples should be drawn. Currently, it searches the correlator path for any files starting with `sonia` or `ponia`, depending on the provided channel.
- `get_channel` – This function opens the file from the valid selection list and reads the correlator data, returning it as a `numpy` array. Currently, it steps along the file according to the channel and extracts the section of the file associated with said channel.

7.2 `feature_extraction.py`

This is the utility script for feature extraction (i.e. attempting to fit the ground state peak).
The main function for the script fits a Gaussian (or other function, if provided) to the spectrum about a feature point and returns the mass and width. If no feature point is specified but a channel is provided, the script will infer the feature point from the effective mass of the channel correlator.
Minimum working examples for the feature extraction script are shown in `feature_extraction.ipynb`.

[4] This is the approach used in the HLT paper, which I am assuming is correct

7.3 `reconstruct_spectrum.ipynb`

This notebook contains some minimum working examples for combining the Backus-Gilbert coefficients with the Gen-2L correlators. This notebook must be in the same directory as the `backus_utils.py` file, but should work mostly out-of-the-box (provided `gvar` has been installed). The `path_to_coeffs` and `path_to_gen2l` variables should be set according to their locations in your filesystem.

7.4 `feature_extraction.ipynb`

This notebook contains the minimum working examples for the feature extraction script. This notebook must be in the same directory as the `feature_extraction.py` file. The `path_to_coeffs` and `path_to_gen2l` variables should be set according to their locations in your filesystem.

7.5 `HMR vs Tikhonov-Laplace.ipynb`

This notebook was used for the HMR vs Tikhonov coefficient analysis. This notebook requires coefficients from the `hmr` branch to be generated for multiple α values using both covariance and Tikhonov regularisation.

A `bgv6_spread.c` Coefficients

The coefficients corresponding to the spread criterion are given by minimising the width measure

$$W_1[c_\tau] = 12\int_{\omega_{\min}}^{\omega_{\max}} (\omega-\omega_0)^2 A(\omega,\omega_0)^2 d\omega - \lambda\left[\int_{\omega_{\min}}^{\omega_{\max}} A(\omega,\omega_0)d\omega - 1\right]. \tag{13}$$

Differentiating with respect to $c_{\tau'}$ yields

$$\partial_{c_{\tau'}} W_1[c_\tau] = \sum_\tau 24\int_{\omega_{\min}}^{\omega_{\max}} (\omega-\omega_0)^2 c_\tau e^{-\omega(\tau+\tau')}d\omega - \lambda\int_{\omega_{\min}}^{\omega_{\max}} e^{-\omega\tau'}d\omega, \tag{14}$$

at which point it is now obvious that the area constraining term is required to remove the trivial minimisation $c_\tau = 0 \ \ \forall\tau$.
The minimised width measure, along with the area constraint, may be expressed in matrix-vector form by defining the following quantities:

$$W_{\tau\tau'}[\omega_0] = 24\int_{\omega_{\min}}^{\omega_{\max}} (\omega-\omega_0)^2 e^{-\omega(\tau+\tau')}d\omega \tag{15}$$

$$C_{\tau'} = \int_{\omega_{\min}}^{\omega_{\max}} e^{-\omega\tau'}d\omega. \tag{16}$$

The quantity $W_{\tau\tau'}$ is called the kernel width matrix and the quantity $C_{\tau'}$ is the constraint vector.
Using these quantities, the coefficients which simultaneously minimise the width whilst also satisfying the area condition obey

$$\begin{pmatrix} W_{\tau\tau'} & C_\tau \\ C_{\tau'} & 0 \end{pmatrix}\cdot\begin{pmatrix} c_\tau \\ \lambda \end{pmatrix} = \begin{pmatrix} \underline{\mathbf{0}} \\ 1 \end{pmatrix} \tag{17}$$

.
The block matrix on the left hand side of Eq. (17) must be inverted for every sampling point ω_0. Although the block matrix can be inverted analytically,

$$\begin{pmatrix} W_{\tau\tau'} & C_\tau \\ C_{\tau'} & 0 \end{pmatrix}^{-1} = \begin{pmatrix} \underline{\mathbf{0}} & \dfrac{\sum_{\tau'} W^{-1}_{\tau\tau'} C_{\tau'}}{\sum_{\tau\tau'} C_\tau W^{-1}_{\tau\tau'} C'_\tau} \\ \dfrac{\sum_\tau W^{-1}_{\tau\tau'} C_\tau}{\sum_{\tau\tau'} C_\tau W^{-1}_{\tau\tau'} C'_\tau} & \dfrac{-1}{\sum_{\tau\tau'} C_\tau W^{-1}_{\tau\tau'} C'_\tau} \end{pmatrix}, \tag{18}$$

the inverse of the kernel width matrix $W_{\tau\tau'}$ must be calculated by other means.
Using Eq. (18) we can see that the coefficients which minimise the width are given by

$$c_\tau(\omega_0) = \frac{\sum_{\tau'} W^{-1}_{\tau\tau'} C_{\tau'}}{\sum_{\tau\tau'} C_\tau W^{-1}_{\tau\tau'} C'_\tau}. \tag{19}$$

B `bgv6_leastsq.c` Coefficients

The coefficients corresponding to the least-squares criterion are given by minimising the width measure

$$W_2[c_\tau] = \int_{\omega_{\min}}^{\omega_{\max}} \left[A(\omega, \omega_0) - \delta(\omega - \omega_0)\right]^2 d\omega. \tag{20}$$

Differentiating with respect to $c_{\tau'}$ yields

$$\partial_{c_{\tau'}} W_2[c_\tau] = 2\left[\sum_\tau \int_{\omega_{\min}}^{\omega_{\max}} c_\tau e^{-\omega(\tau+\tau')} d\omega\right] - 2e^{-\omega_0\tau'}, \tag{21}$$

whose minimum can be expressed by the simple matrix-vector product

$$\sum_{\tau'} c_\tau W_{\tau\tau'} = C_{\tau'}, \tag{22}$$

where we have defined

$$W_{\tau\tau'}[\omega_0] = \int_{\omega_{\min}}^{\omega_{\max}} e^{-\omega(\tau+\tau')} d\omega \tag{23}$$

$$C_{\tau'} = e^{-\omega_0\tau'}. \tag{24}$$

The coefficients which minimise the width can then be expressed using a single matrix inverse,

$$c_\tau(\omega_0) = \sum_{\tau'} W^{-1}_{\tau\tau'} C'_\tau. \tag{25}$$

C `bgv6_hmr.c` Coefficients

The coefficients corresponding to the area-constrained, least-squares criterion are given by minimising the width measure

$$W_2[c_\tau] = \int_{\omega_{\min}}^{\omega_{\max}} \left[A(\omega,\omega_0) - \delta(\omega-\omega_0)\right]^2 d\omega - \lambda\left[\int_{\omega_{\min}}^{\omega_{\max}} A(\omega,\omega_0)d\omega - 1\right]. \tag{26}$$

Differentiating with respect to $c_{\tau'}$ yields

$$\partial_{c_{\tau'}} W_2[c_\tau] = 2\left[\sum_\tau \int_{\omega_{\min}}^{\omega_{\max}} c_\tau e^{-\omega(\tau+\tau')}d\omega\right] - 2e^{-\omega_0\tau'} - \lambda\int_{\omega_{\min}}^{\omega_{\max}} e^{-\omega\tau'}. \tag{27}$$

As before, we can express the above via a series of matrix-vector products by defining the quantities

$$W_{\tau\tau'}[\omega_0] = \int_{\omega_{\min}}^{\omega_{\max}} e^{-\omega(\tau+\tau')}d\omega \tag{28}$$

$$C_{\tau'} = e^{-\omega_0\tau'}. \tag{29}$$

$$R_{\tau'} = \int_{\omega_{\min}}^{\omega_{\max}} e^{-\omega\tau'}d\omega \tag{30}$$

such that the minimised width becomes

$$\sum_\tau c_\tau W_{\tau\tau'} = C_{\tau'} + \widetilde{\lambda}R_{\tau'}, \tag{31}$$

where for simplicity we define $\lambda = 2\widetilde{\lambda}$. This condition, along with the area constraint, may be expressed together using

$$\begin{pmatrix} W_{\tau\tau'} & R_\tau \\ R_{\tau'} & 0 \end{pmatrix}\begin{pmatrix} c_\tau \\ \widetilde{\lambda} \end{pmatrix} = \begin{pmatrix} C_{\tau'} \\ 1 \end{pmatrix}. \tag{32}$$

Inverting the block-matrix in Eq. (32) is analytically more involved than in the case of the spread criterion, given by

$$\begin{pmatrix} W_{\tau\tau'} & R_\tau \\ R_{\tau'} & 0 \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{W}^{-1}\left[\mathbb{1} - \dfrac{\boldsymbol{R}\boldsymbol{W}^{-1}}{\boldsymbol{R}^T\boldsymbol{W}^{-1}\boldsymbol{R}}\right] & \dfrac{\boldsymbol{W}^{-1}\boldsymbol{R}}{\boldsymbol{R}^T\boldsymbol{W}^{-1}\boldsymbol{R}} \\ \dfrac{\boldsymbol{R}^T\boldsymbol{W}^{-1}}{\boldsymbol{R}^T\boldsymbol{W}^{-1}\boldsymbol{R}} & \dfrac{1}{\boldsymbol{R}^T\boldsymbol{W}^{-1}\boldsymbol{R}} \end{pmatrix}, \tag{33}$$

where for clarity we have used $\boldsymbol{W} = W_{\tau\tau'}$, $\boldsymbol{R} = R_\tau$ and $\boldsymbol{R}^T = R_{\tau'}$. The coefficients which then minimise both the width and the area constraint simultaneously are given by

$$c_\tau(\omega_0) = \sum_{\tau'}\left\{W^{-1}_{\tau\tau'}C_{\tau'} + W^{-1}_{\tau\tau'}R_{\tau'}\left[\frac{1-\sum_{\tau\tau'}R_\tau W^{-1}_{\tau\tau'}C_{\tau'}}{\sum_{\tau\tau'}R_\tau W^{-1}_{\tau\tau'}R_{\tau'}}\right]\right\}. \tag{34}$$