WDW.GC - User Manual

Hamilton José Brumatto

April 11, 2025

Contents

1 Downloading and running the program **5**

2 Normal use **7**

2.1 The problem . . . 7
2.2 Defining the parameters . . . 7
2.3 Running the program . . . 9
2.4 The recovery points . . . 10
2.5 Restrictions . . . 11

3 Advanced use **13**

3.1 Changing effective potential . . . 13
3.2 Changing INI parameters . . . 14
3.3 Changing Ψ_0 . . . 14

4 BLAS library **17**

4.1 Macros . . . 17
4.2 Classes: `sm_matrix<T>` . . . 17
4.2.1 Description: . . . 17
4.2.2 Public methods . . . 18
4.2.3 Library functions . . . 19

Chapter 1

Downloading and running the program

wdw.gc program can be found at: `https://github.com/brumatto/wdw.gc.git`, or
`gh repo clone brumatto/wdw.gc`

The files are:

SMmatrix.hpp BLAS routines for linear algebra operations when using Conjugate Gradient.

GCsolve.hpp Conjugate Gradient library to solve a linear system (header file: `GCsolve.hpp`)

wdw.gc.aux.cpp Auxiliary functions to initialize wdw.gc parameters (header file: `wdw.gc.hpp`)

wdw.gc.cpp Main program to read ini file with parameters and solve the Wheeler DeWitt equation as a Initial Value Problem (header file: `wdw.gc.hpp`).

wdw.ini A sample ini file with parameters of one test case

compile A script file to compile the source code and link edit it with OpenMP using GNU C++ 17 compiler.

After downloading just compile it with GNU C++ 17 or higher with OpenMP option or run the script file `compile` to do it. The binary code is now ready to use.

Chapter 2

Normal use

2.1 The problem

The wdw.gc program is meant to solve the Wheeler-DeWitt equation:

$$\left[-\frac{1}{12}\frac{\partial^2}{\partial a^2}+\frac{1}{2}\frac{\partial^2}{\partial \varphi^2}+V_{eff}(a,\varphi)\right]\Psi(a,\varphi,\tau)=i\frac{\partial}{\partial \tau}\Psi(a,\varphi,\tau)$$

where a is the scalar factor for the universe, φ is a scalar field, τ is the dynamic variable associated with the perfect fluid based on radiation, V_{eff} the effective potential that provide a barrier for the wave function and Ψ the universe wave function. The Effective Potencial used is:

$$V_{eff}(a,\varphi)=3a^2-\Lambda a^4-\frac{1}{2}\varphi^2-\frac{1}{2}m^2a^2\varphi^2$$

where Λ is the Einstein Universe Constant representing de radiation vacuum, and m the mass conformally coupled with the radiation.

For solving the equation the boundary conditions are defined:

$$\begin{cases}\Psi(a=0,\varphi,\tau)=0,\\ \Psi(a=\infty,\varphi,\tau)=0,\\ \Psi(a,\varphi=-\infty,\tau)=0,\\ \Psi(a,\varphi=\infty,\tau)=0\end{cases}$$

and the initial condition for $\Psi(a,\varphi,0)$ is:

$$\Psi(a,\varphi)=\frac{2^{15/4}}{\sqrt{\pi}}E_a^{3/4}E_\varphi^{1/4}a\exp(-4E_a a^2-2E_\varphi\varphi^2)$$

where E_a and E_φ are de energy associate to the scalar factor and scalar field respectively.

2.2 Defining the parameters

The previous section shows a list o parameters whose value will determine the behavior of the wave function of the universe when evolving from the initial condition. Those parameters are:

Λ (Lambda), m (m), E_a (Ea) and E_φ (Ephi). These parameters can be set to a value for each case test in an INI file.

By default `wdw.gc` looks for `wdw.ini` at current directory, but it is possible to designate any path to a valid ini file on a command line:

```
wdw.gc <ini file>
```

Besides these parameters the ini file allows the user to stablish various other parameters. The list bellow describe the parameters, and the specific name for each parameter comes between parentheses:

- The numerical infinity for the scalar factor range and scalar field range: a_∞ (a) and $\pm\varphi_\infty$ (phi)
- The number of points in both direction por Crank Nicolson finite differences: N_a (Na) and N_φ (Nphi). This will determine the finite differences: $\Delta a = a_\infty/N_a$ and $\Delta\varphi = 2\varphi_\infty/N_\varphi$.
- The interval $\Delta\tau$ (DeltaT) for each iteration on τ evolution. And the last value for τ (Tlast) to stop evolution.
- We create a recovery point after a number of $\Delta\tau$ evolving, these is designate at a parameter "(Print)". So it is always possible to continue after some point. This recovery point is associated with a "resume time" (rest) as parameter.
- We define path (BaseDir) and prefix (file) for files of the recovery point. Each file created will be created with the value of τ in a format "00.000" at it name. Although we specify a "rest" parameter it is also necessary to specify the name of the resume file (resfile).
- A resume parameter (Resume) is need to indicate if the program will run from start or resume from some recovery point.
- It is useful to define the "Tunneling and Circumventing Surface" the value for a where the Effective potential has it maximum peak: $a_{V_{max}}$ (aVmax)
- And for the Conjugate Gradiente algorithm it is necessary to specify de maximum number of iterations (to stop the program if there is no convergence) i_{max} (Imax) and de maximum error on the norm of Ψ to accept the value as correct: $e_{max}(Emax)$
- We also allow the user to define how many threads will be created to run the program (NProc). If this parameter is not set than the program will use the maximum the hardware offers.

The table 2.1 shows an exemple of ini file. Any line beginning with # or any blank line is discarded. The program expects to read a line containing a pair: (parameter,value). The parameter is case sensitive, and in this example we can see a completeness initialization of the parameters

```
# Comments line beginning with # and blank lines are ignored.
# Each line has a variable (CASE SENSITIVE) and its value.
# Any variable not found here will be assigned a default value.
# Variable not needed is ignored

# Parallelism
# NProc 40

# Cranck-Nicholson:
Na 6000
Nphi 9000
a 20
phi 40

# Vef:
Lambda 1.5
m 0.1
aVmax 1.0

# Psi0
Ea 5.498
Ephi 4.0

# GC iterations
Imax 500
Emax 1e-8
Resume false

# Time evolution
Tlast 1.0
DeltaT 0.00001
Print 2500

# Point of recovery
rest 0.755
resfile a5phi4_00.755
file a5phi4_
BaseDir lbd1.5/m0.1/a5phi4

# Other programs:  Convertdat
Zero 1e-5
```

Table 2.1: INI file sample

2.3 Running the program

The program reads the ini file looking for the variable names to set its value. When a variable is not found, the value is set to a default value, but not ever this value has a meaningfull definition. As an special case, `NProc`, the number of threads, when not set will be used the maximum the hardware offers.

It is possible to define other parameters, that is not used by wdw.gc program. It will be ignored. This is the case when we create an auxiliary code that have to deal with the data recorded and also needs part of these parameters to properly open and process the file, and uses others parameters necessary, an exemple is the parameter "`Zero`", used for a file to convert form binary data to text data, useful to draw graphics.

It is possible to run many test cases just changing the `BaseDir` or the prefix for the file name: `file`

When running, the program will generate the following output:

- Default console: a list of mean values for: Ψ^2, $\langle a\rangle$, $\langle\varphi\rangle$ and $\langle\varphi^2\rangle$. Each line starts with the value of τ and then each of the mean values in 4 ranges: full - double integration in Δa and $\Delta\varphi$ for all range of them, Integration only after "Tunneling and Circumventing surface" (TCs), only before TCs and only inside the effective potencial.

- Error console: Initially it shows the initializing steps, when it finishes the calculation of matrices and vectors. Then when it begins the Conjugate Gradient algorithm it prints each iteration for the algorithm, printing the norm of Ψ e maximum between Ψ_i and Ψ_{i-1}, and the maximum value for Ψ.

2.4 The recovery points

The path indicated by `BaseDir` parameter, from local directory, will be recorded all data as recovery points, even the first point. The points are recorded based upon `DeltaT` and `Print` parameters, the `DeltaT` indicates the step for the evolution parameter τ and after `Print` steps the state of the wave function Ψ is recorded.

The file recorded has a prefix name defined by the parameter `file` followed by a temporal position defined by τ in de format "##.###" followed by the suffix ".dat". Considering de example on table 2.1 the first 4 recovery points recorded are {`a5phi4_00.000.dat, a5phi4_00.025.dat, a5phi4_00.050.dat, a5phi4_00.075.dat`} corresponding respectively to $\tau = 0.0, \tau = 0.025, \tau = 0.05$ and $\tau = 0.075$.

Each file is recorded as $(N_a \times N_\varphi) + 1$ pair of doubles in the binary format, each pair correspond to the real and imaginary value of Ψ in the point of a grid. The parameters `Na` and `Nphi` define the size of the grid and the parameters `a` corresponding to a_∞ and `phi` to φ_∞ define the range for the values. So the space between each point: $\Delta a = a_\infty/N_a$ and $\Delta\varphi = 2\varphi_\infty/N_\varphi$ allows us to define a and φ position for each Ψ recorded:

$$\begin{aligned}
&\Psi_{i,j} = \Psi(a_i, \varphi_j)\\
&a_i = a_0 + i\Delta a; \varphi_j = \varphi_0 + j\Delta j\\
&a_0 = 0; a_{N_a} = a_\infty; \varphi_0 = -\varphi_\infty; \varphi_{N_\varphi} = \varphi_\infty
\end{aligned}$$

The sequence recorded is: $\Psi_{0,0}, \Psi_{0,1}, \Psi_{0,2}, \ldots, \Psi_{0,N_\varphi}, \Psi_{1,0}, \ldots \Psi_{N_a,N_\varphi}$.

The recovery points are used as Ψ state at each evolution in τ for posterior analysis on a cosmological study and as a recovery point to resume running the program after unexpected stop. The parameter `Resume` if false defines that the program will start from $\Psi(\tau = 0)$ as in initial condition, if true it defines that the program would continue form t indicated by the parameter `rest` and will load Ψ from the file indicated by the parameter `resfile` located at `BaseDir` path.

2.5 Restrictions

This program was made to use on a HPC (High Performance Cluster), its speedup shows that the best performance occurs using 30 50 threads. The main restriction to the use is the memory size.

Running the program it is created about 25 “vectors” of size $N_a \times N_\varphi$ of complex values (two doubles). For $N_a \times N_\varphi \approx 10^{10}$ it is necessary about 128GBytes of memory RAM. It is not convenient to use with a grid over this size.

It is not possible to change the boundary conditions of the Differential Partial Equation. We choose Psi for $a = 0$, a_∞ and $\pm\phi_\infty$ is defined as $\Psi = 0$ and it is intrinsic defined on the code. To use something like $\left.\dfrac{\partial \Psi}{\partial a}\right|_{a=0} = 0$ or on the other border it will be necessary develop another code.

Chapter 3

Advanced use

3.1 Changing effective potential

We defined the effective potential as:

$$V_{eff}(a, \varphi) = 3a^2 - \Lambda a^4 - \frac{1}{2}\varphi^2 - \frac{1}{2}m^2 a^2 \varphi^2$$

But it is possible for the Wheeler-DeWitt equation to specify a different potential, let us suppose a new potential as:

$$V_{eff}(a, \varphi) = 3a^2 - \Lambda a^4 + \frac{aV_0}{\cosh a} - \frac{1}{2}\varphi^2 - \frac{1}{2}m^2 a^2 \varphi^2$$

Although our program was built to the first potential it is easy to change the code to choose another one.

The code `wdw.gc.cpp` has on its lines 24 through 28, and on the line 112, the following code:

```
 24  double potential(double a, double phi, double lambda, double mass) {
 25       double a2 = a*a;
 26       dobule phi2 = phi*phi;
 27       return (3*a2 - lambda*a2*a2 - phi2/2 - mass*mass*a2*phi2/2);
 28  }
  ⋮
112            Vef[j*(Na+1)+i] = potential(ai,phij,lambda,mass);]
```

on line 112, the variables `ai` and `phij` represent respectively a_i and φ_j for the point on de grid. To change the effective potential on the code we just need to change the function properly on lines 24 through 28 and how it is called on line 112.

Sometimes it is necessary to add some new parameter as V_0 this is done as explained in section 3.2.

After changing the potential to the new one one have only to recompile the code using the script `compile`, or manually with gnu g++ 17 or higher. Do not forget to specify OpenMP option to compile and link.

3.2 Changing INI parameters

On the last section we need to change the effective potential and the new one needs a new parameter: V_0, so it is necessary to include this parameter on the code and on the INI file.

First, it is necessary to define the parameter name for the INI file: let us say: "V0". So just include the pair "V0 ¡value¿" in the file, let us do it near other values we defined por the potential as on the example bellow:

```
⋮
# Vef:
Lambda 1.5
m 0.1
aVmax 1.0
V0 1.0
⋮
```

On the code it is necessary to create a variable, let us say "`double v0`", line 34 on code `wdw.gc.cpp` already has the declaration of the variables `lambda` and `mass`, it is convenient to add this one on that line:

```
24  double lambda, mass, v0;
```

Lines 58 through 85 has the code load the ini values, just add the new one as the example bellow:

```
  ⋮
65  // Vef
66  else if (variavel == "Lambda") iss >> lambda;
67  else if (variavel == "m") iss >> mass;
68  else if (variavel == "aVmax") iss >> avmax;
69  else if (variavel == "V0") iss >> v0;
  ⋮
```

After that just recompile the code using the script `compile` or manually with gnu g++ 17 or higher. Do not forget to specify OpenMP option to compile and link.

3.3 Changing Ψ_0

The initial condition Ψ_0 is based on a first choice to a wave package form:

$$\Psi_0 = Caexp{-}Aa^2 - B\varphi^2 \tag{3.1}$$

The constants involved: A, B and C are obtained after considering the restrictions:

$$\int_{-\infty}^{\infty}\int_{0}^{\infty} \Psi^*\Psi da d\varphi = 1$$
$$\int_{-\infty}^{\infty}\int_{0}^{\infty} \Psi^*\frac{p_a^2}{12}\Psi da d\varphi = E_a$$
$$\int_{-\infty}^{\infty}\int_{0}^{\infty} \Psi^*\frac{p_\varphi^2}{2}\Psi da d\varphi = E_\varphi$$

where p_a and p_φ are the canonical momentum respectively of scalar factor and scalar field.

Let us suppose we change, if physically possible the wave package to:

$$\Psi_0 = Ca^2 exp(-Aa^2 - B\varphi^2) \tag{3.2}$$

then the constants A, B, C will change and we have to deal of changing the initial condition. This is made directly on code. The file `wdw.gc.aux.cpp` has the implementation of the function: `createPsi0` there we must code the new initial condition Ψ_0. If it is necessary to change the function protocol (definition of the function), so it is necessary to change it on the header file `wdw.gc.hpp` and change its call in the main code at `wdw.gc.cpp` at line 124:

```
124  psi = createPsi0(Na,Nphi,inftya,inftyphi,Ea,Ephi);
```

Chapter 4

BLAS library

Among the files it is included the header file `SMmatrix.hpp` it is a BLAS (Basic Linear Algebra Subprograms) library developed to work with sparse matrices. It is defined by a class whose values are template class. Here we have the methods, functions and macros defined to work with this library.

4.1 Macros

`#define ZERO (1e-300)`

We consider a zero value to a value whose absolute form is less or iqual 10^{-300}. These happens because we cannot compare float point values due to its representation on a limited binary representation. Also it is important when using complex values.

4.2 Classes: `sm_matrix<T>`

4.2.1 Description:

`template<Class T>`

`class sm_matrix<T>`

A cell of the matrix is a `pair<int,T>`, each cell carries its index 'j' as the first element of the pair, each cell also contains an associated value, as the second element of the pair. If the value of the cell is 0 (¡ ZERO in absolute value), it must be removed from the structure. A vector (`vector<pair<int,T>>`) of cells is a row of the matrix.

The `sm_matrix` structure is constructed with a row vector, where each row is the cell vector (`vector<vector<pair<int,T>>>`). A cell (i,j) of the matrix has its index 'i' (row) as the index of the row vector. The index 'j' of the cell (column) is obtained inside the cell, in the cell vector that forms a row.

4.2.2 Public methods

	`sm_matrix(int n)`
Description:	Constructor, initializes a sparse matrix of dimension n with n rows where each row is an empty vector
Parameters:	**n**: the dimension of the sparse square matrix

	`int size()`
Description:	Returns the size n of the sparse square matrix $n \times n$
Returns:	The matrix dimension

	`void setCell(int i, int j, T value)`
Description:	Assigns the value `value` to cell (`i`,`j`) of the matrix
Parameters:	**i**: the index of row `i` of cell (i,j)
	j: the index of column `j` of cell (i,j)
	value: the value of cell (i,j), if the value is `0 (< ZERO)` the cell will be deleted.

	`T getCell(int i,int j)`
Description:	returns the value of cell (`i`,`j`) of the matrix
Parameters:	**i**: the index of row `i` of cell (i,j)
	j: the index of column `j` of cell (i,j)
Returns:	the value of cell (i,j), or 0 if it is not in the matrix

	`void swapLine(int i, int j)`
Description:	swap lines `i` and `j` of matrix
Parameters:	**i**: one of the lines of the exchange
	j: the other line of the exchange

	`vector<pair<int,T>> line(int i)`
Description:	returns a vector with the elements of row i
Parameters:	**i**: the index of the row to be returned
Returns:	a vector containing the cells of the row

	`void setline(int i, vector<pair<int,T>> line)`
Description:	assigns a vector of cells to a row
Parameters:	**i**: the index of the line that will be initialized
	line: a vector of cells that will be initialized as a row

`vector<pair<int,T>>::iterator begin(int i)`	
Description:	returns the starting *iterator* for the cells in a row
Parameters:	**i**: the line index
Returns:	an iterator to manipulate the cells of a row, pointing to the beginning

`vector<pair<int,T>>::iterator end(int i)`	
Description:	returns the final *iterator* for the cells in a row
Parameters:	**i**: the line index
Returns:	an iterator to manipulate the cells of a row pointing to the end

4.2.3 Library functions

`vector<T> sm_vmultMatrix(sm_matrix<T> &m, vector<T> &b)`
Takes the product of a $n \times n$ matrix by a vector of size n

Parameters

`m` a sparse matrix.
`b` a vector.

Returns

A vector resulting from the product .

`sm_matrix<T> sm_mmultMatrix( sm_matrix<T> &a, sm_matrix<T> &b)`
mmultMatrix returns a matrix that is the product of the two passed matrices

Parameters

`a` a sparse matrix nxn.
`b` a sparse matrix nxn.

Returns

the product of the matrices.

`sm_matrix<T> sm_invertMatrix(sm_matrix<T> &m)`
nvertMatrix returns the inverse matrix. When the identity column is null, a row swap is made to the row whose column is not null.

Parameters

`m`, a sparse matrix nxn.

Returns

the inverse of the matrix.

`vector<T> sm_gaussSolve(sm_matrix<T> &m, vector<T> &b)`
sm_gaussSolve solves the linear system by gauss elminiação (Gauss-Seidel) mx = b, returning x.

Parameters

`m` a sparse matrix nxn.
`b` a n sized vector.

Returns

a solution vector.

`int sm_GJsolve(sm_matrix<T> &m, vector<T> &x, vector<T> &b, int max,`
`double &e)`
GJsolve is a solver of linear system mx=b by the Gauss-Jacobi method, returning in x the solution

Parameters

`m`, a sparse matrix nxn.
`x`, a vector of size n with the initial estimate of the iteration. At the end you will have the vector closest to the solution
`b`, a vector of size n with the system font.
`max`, is the maximum number of iterations, if the system does not converge quickly.
`e`, is the maximum expected approximation for the values of x, we want: `abs(x(k) - x(k-1)) < e`, for all x, the worst value of the approximation is returned in this variable.

Returns

will return the maximum number of iterations reached .