# SMCSolver: Structured Markov Chains Solver

Dario A. Bini, Beatrice Meini
and Sergio Steffé
Dipartimento di Matematica
Università di Pisa

Pisa, Italy

Email: bini, meini, steffe @dm.unipi.it

November 29, 2006

**Abstract**

Extended documentation of the package SMCSolver (Structured Markov Chains Solver) Version 1.2 .

## 1 Introduction

The package Structured Markov Chains Solver (SMCSolver) has the purpose of solving the most important classes of Markov chains encountered in queueing models.

This package implements the most advanced available algorithms for QBD, M/G/1 and G/M/1 problems.

The algorithms have been implemented as a Fortran 95 library.

A user-friendly graphical interface which allows an easy use of the software tool is provided. The interface is written in C and relies on the GTK graphic libraries, and runs on a Linux Workstation.

The package has been developed as part of a joint research collaboration with

> Benny Van Houdt
> Department of Mathematics and Computer Science
> University of Antwerp
> Antwerpen, Belgium

who wrote a Matlab ToolBox with similar functionalities.

The joint work (Version 1.1) [8] [9] has been presented at the workshop: SMCTOOLS: Tools for solving Structured Markov Chains

organized within the:
VALUETOOLS 2006 CONFERENCE, held in Pisa, Italy - October 10, 2006.

The Fortran code includes some code previously released by Bini and Meini, and available at http://www.netlib.org, plus new routines. The graphic interface has been written from scratch for SMCSolver.

The MATLAB TOOLBOX is available at:

http://www.win.ua.ac.be/~ vanhoudt/

and SMCSOLVER is available at:

http://bezout.dm.unipi.it/SMCSolver/

# 2 The problems

We consider row stochastic matrices $P$ which can be partitioned into blocks $P_{i,j}$. The matrix $P$ is semi-infinite, i.e., its blocks $P_{i,j}$ have subscripts $i, j \in \mathbb{N}$. In this framework, the main computational problem is computing the *invariant probability vector*, i.e., the infinite nonnegative row vector $\boldsymbol{\pi}$ such that $\boldsymbol{\pi}P = \boldsymbol{\pi}$, and $\boldsymbol{\pi}\boldsymbol{e} = 1$. Here $\boldsymbol{e}$ is the vector with components equal to 1. Throughout the paper we assume $P$ irreducible.

Due to the block structure of $P$, it is convenient to partition the vector $\boldsymbol{\pi}$ into subvectors $\boldsymbol{\pi}_i$, $i \in \mathbb{N}$. According to the specific structure of $P$ we may classify the Markov chains into suitable classes. In this section we refer the reader to the books [20, 21, 16, 4].

## 2.1 QBD Markov chains

QBD Markov chains are defined by the transition matrix

$$P = \begin{bmatrix} B_0 & B_1 & & & 0 \\ B_{-1} & A_0 & A_1 & & \\ & A_{-1} & A_0 & A_1 & \\ & & A_{-1} & A_0 & \ddots \\ 0 & & & \ddots & \ddots \end{bmatrix}, \tag{1}$$

where $A_{-1}, A_0, A_1 \in \mathbb{R}^{m\times m}$ and $B_{-1} \in \mathbb{R}^{m\times n}$, $B_0 \in \mathbb{R}^{n\times n}$,, $B_1 \in \mathbb{R}^{n\times m}$, are nonnegative matrices such that $[B_0, B_1]$ , $[B_{-1}, A_0, A_1]$ and $[A_{-1}, A_0, A_1]$ are row stochastic. Assume that $A = A_{-1} + A_0 + A_1$ is irreducible. The drift of a QBD Markov chain is defined by $\mu = \boldsymbol{\alpha}^{\mathrm{T}}(-A_{-1} + A_1)\boldsymbol{e}$, where $\boldsymbol{\alpha}$ is the stationary probability vector of $A$. We recall that a QBD is positive recurrent if $\mu < 0$, null recurrent if $\mu = 0$ and transient if $\mu > 0$.

Define $G$, $R$ and $U$ the minimal nonnegative solutions of the matrix equations

$$\begin{aligned} G &= A_{-1} + A_0 G + A_1 G^2, \\ R &= A_1 + R A_0 + R^2 A_{-1}, \\ U &= A_0 + A_1 (I - U)^{-1} A_{-1}. \end{aligned} \tag{2}$$

One has $G = (I-U)^{-1}A_{-1}$, $R = A_1(I-U)^{-1}$, moreover, if the QBD is positive recurrent, and if $B_1 = A_1$, it holds

$$\begin{aligned} &\boldsymbol{\pi}_k = \boldsymbol{\pi}_0 R^k, \qquad \text{for } k \geq 0, \\ &\boldsymbol{\pi}_0(B_0 + B_{-1}G) = \boldsymbol{\pi}_0, \\ &\boldsymbol{\pi}_0(I-R)^{-1}\boldsymbol{e} = 1. \end{aligned} \tag{3}$$

## 2.2 M/G/1-type Markov chains

M/G/1-type Markov chains are defined by the transition matrix

$$P = \begin{bmatrix} B_0 & B_1 & B_2 & B_3 & \dots \\ B_{-1} & A_0 & A_1 & A_2 & \dots \\ & A_{-1} & A_0 & A_1 & \ddots \\ & & A_{-1} & A_0 & \ddots \\ 0 & & & \ddots & \ddots \end{bmatrix}, \tag{4}$$

where $A_i$, for $i \geq -1$ are nonnegative matrices in $\mathbb{R}^{m\times m}$, $B_0$ is a nonnegative matrix in $\mathbb{R}^{n\times n}$, $B_{-1}$ is a nonnegative matrix in $\mathbb{R}^{n\times m}$, and $B_i$, for $i \geq 1$ are nonnegative matrices in $\mathbb{R}^{m\times n}$, such that $[B_0, B_1, B_2, \dots]$, $[B_{-1}, A_0, A_1, A_2, \dots]$, $[A_{-1}, A_0, A_1, A_2, \dots]$, are row stochastic. Throughout we assume $A = \sum_{i=-1}^{+\infty} A_i$ irreducible. The drift of an M/G/1-type Markov chain is defined by $\mu = \boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{a}$, where $\boldsymbol{\alpha}$ is the stationary probability vector of $A$ and $\boldsymbol{a} = \sum_{i=-1}^{+\infty} iA_i\boldsymbol{e}$. We recall that a Markov chain is recurrent iff $\mu \leq 0$, positive recurrent iff $\mu < 0$ and $\boldsymbol{b} = \sum_{i=1}^{+\infty} iB_i\boldsymbol{e} < \infty$, transient iff $\mu > 0$, null recurrent iff either $\mu = 0$ or $\mu < 0$ and $\sum_{i=-1}^{+\infty} iB_i\boldsymbol{e} = \infty$.

Define $G$ the minimal nonnegative solution of the matrix equation

$$G = \sum_{i=-1}^{+\infty} A_i G^{i+1}. \tag{5}$$

In the applications, the infinite sequence of data $A_i$, $i = -1, 0, 1, 2, \dots$, is truncated to the finite size $K$ such that $\sum_{i=-1}^{K} A_i$ is numerically stochastic. In this way, the equation (5) turns into

$$G = \sum_{i=-1}^{K} A_i G^{i+1}. \tag{6}$$

If the Markov chain is positive recurrent, and $B_{-1} = A_{-1}$ the following recursive formula due to Ramaswami [22] holds

$$\boldsymbol{\pi}_k = \left(\boldsymbol{\pi}_0 B_k^* + \sum_{i=1}^{k-1} \boldsymbol{\pi}_i A_{k-i}^*\right)(I - A_0^*)^{-1}, \qquad \text{for } k \geq 1, \tag{7}$$

where

$$A_k^* = \sum_{i=k}^{+\infty} A_i G^{i-k}, \qquad B_k^* = \sum_{i=k}^{+\infty} B_i G^{i-k}, \qquad \text{for } k \geq 0, \tag{8}$$

and $\boldsymbol{\pi}_0$ is such that

$$\boldsymbol{\pi}_0 B_0^* = \boldsymbol{\pi}_0, \quad \boldsymbol{\pi}_0 \boldsymbol{b} - \mu \boldsymbol{\pi}_0 \boldsymbol{e} + \boldsymbol{\pi}_0 (I - B)(I - A)^{\#} \boldsymbol{a} = -\mu, \tag{9}$$

where $B = \sum_{i=0}^{+\infty} B_i$ and the operator $(\cdot)^{\#}$ denotes the group inverse. A fast version of Ramaswami formula based on FFT is shown in [17]. It outperforms the formula based on (7) and (8) only if a very large number of components $\boldsymbol{\pi}_n$ is required.

## 2.3 G/M/1-type Markov chains

G/M/1-type Markov chains are defined by the transition matrix

$$P = \begin{bmatrix} B_0 & B_1 & & & 0 \\ B_{-1} & A_0 & A_1 & & \\ B_{-2} & A_{-1} & A_0 & A_1 & \\ B_{-3} & A_{-2} & A_{-1} & A_0 & \ddots \\ \vdots & \vdots & \ddots & \ddots & \ddots \end{bmatrix}, \tag{10}$$

where $A_{-i}$, $i \geq -1$ are nonnegative matrices in $\mathbb{R}^{m \times m}$, $B_0$, is a nonnegative matrix in $\mathbb{R}^{n \times n}$, $B_1$, is a nonnegative matrix in $\mathbb{R}^{m \times n}$, and $B_{-i}$, $i \geq 1$ are nonnegative matrices in $\mathbb{R}^{n \times m}$, and $[B_0, B_1]$, $[B_{-1} + A_0 + A_1]$, $[B_{-k}, A_{-k+1}, \ldots, A_0, A_1]$ are row stochastic for all $k \geq 2$.

If $A = \sum_{i=-1}^{+\infty} A_{-i}$ is not stochastic, then the Markov chain is positive recurrent. If $A$ is stochastic then the Markov chain is positive recurrent if $\delta < 0$, null recurrent if $\delta = 0$, and transient if $\delta > 0$, where $\delta = \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{a}$, $\boldsymbol{\alpha}$ is such that $\boldsymbol{\alpha}^{\mathrm{T}} A = \boldsymbol{\alpha}^{\mathrm{T}}$, $\boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{e} = 1$, and $\boldsymbol{a} = \sum_{i=-1}^{+\infty} i A_{-i} \boldsymbol{e}$.

Define $R$ the minimal nonnegative solution of the matrix equation

$$R = \sum_{i=-1}^{+\infty} R^{i+1} A_{-i}. \tag{11}$$

If the Markov chain is positive recurrent, and if $B_1 = A_1$ then

$$\boldsymbol{\pi}_k = \boldsymbol{\pi}_0 R^k \qquad \text{for } k \geq 1, \tag{12}$$

where $\boldsymbol{\pi}_0$ is characterized by the system

$$\boldsymbol{\pi}_0 = \boldsymbol{\pi}_0 \sum_{i=0}^{+\infty} R^i B_{-i}, \qquad \boldsymbol{\pi}_0 (I - R)^{-1} \boldsymbol{e} = 1. \tag{13}$$

It is worth mentioning that, if the matrix $A = \sum_{i=-1}^{+\infty} A_{-i}$ is irreducible and stochastic, the connection between M/G/1 and G/M/1-type Markov chains

is extremely simple. Indeed, define $D = \mathrm{diag}(\boldsymbol{\alpha})$, where $\boldsymbol{\alpha}$ is the strictly positive stationary probability vector of $A$ and define $\widetilde{A}_i = D^{-1}A_{-i}^{\mathrm{T}}D$, for $i = -1, 0, 1, \ldots$. Now, take any solution $R$ of (11) and define $\widetilde{G} = D^{-1}R^{\mathrm{T}}D$. It is easy to verify that $\widetilde{G}$ is a solution of

$$X = \sum_{i=-1}^{+\infty} \widetilde{A}_i X^{i+1}. \tag{14}$$

A consequence of this property is that, to determine $R$ for a positive recurrent G/M/1-type Markov chain is equivalent to determining $G$ for a transient M/G/1-type Markov chain. Conversely, to determine $R$ for a null recurrent or transient G/M/1-type Markov chain is equivalent to determining $G$ for a recurrent M/G/1-type Markov chain.

## 2.4 The case of Non-skip-free processes

Markov chains which are non-skip-free to lower levels are defined by the generalized block upper Hessenberg matrix

$$P = \begin{bmatrix} B_0 & B_1 & B_2 & B_3 & \cdots \\ B_{-1} & A_0 & A_1 & A_2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ B_{-N+1} & A_{-N+2} & A_{-N+3} & A_{-N+4} & \ddots \\ A_{-N} & A_{-N+1} & A_{-N+2} & A_{-N+3} & \ddots \\ & A_{-N} & A_{-N+1} & A_{-N+2} & \ddots \\ & & A_{-N} & A_{-N+1} & \ddots \\ & & & A_{-N} & \ddots \\ 0 & & & & \ddots \end{bmatrix}. \tag{15}$$

for $m \times m$ blocks $A_i$, $i \geq -N$ and $B_i$, $i \geq -N+1$, where $N \geq 1$ is an integer. Markov chains which are non-skip-free to upper levels can be similarly defined in terms of generalized block lower Hessenberg matrix. The matrix $P$ can be reblocked into blocks $\mathcal{B}_i$, $i \geq 0$ and $\mathcal{A}_i$, $i \geq -1$ of size $mN$ as

$$P = \begin{bmatrix} \mathcal{B}_0 & \mathcal{B}_1 & \mathcal{B}_2 & \mathcal{B}_3 & \cdots \\ \mathcal{A}_{-1} & \mathcal{A}_0 & \mathcal{A}_1 & \mathcal{A}_2 & \cdots \\ & \mathcal{A}_{-1} & \mathcal{A}_0 & \mathcal{A}_1 & \ddots \\ & & \mathcal{A}_{-1} & \mathcal{A}_0 & \ddots \\ & & & \ddots & \ddots \end{bmatrix}. \tag{16}$$

Therefore, in principle, it can be solved like a standard M/G/1 Markov chain. In this case, the solution $\mathcal{G}$ of the equation $\mathcal{G} = \sum_{i=-1}^{+\infty} \mathcal{A}_i \mathcal{G}^{i+1}$ can be written

as

$$\mathcal{G} = U^{-1}L \tag{17}$$

where

$$\begin{aligned} U &= \begin{bmatrix} I & & & 0 \\ -G_1 & I & & \\ \vdots & \ddots & \ddots & \\ -G_{N-1} & \dots & -G_1 & I \end{bmatrix}, \\ L &= \begin{bmatrix} G_N & G_{N-1} & \dots & G_1 \\ & \ddots & \ddots & \vdots \\ & & G_N & G_{N-1} \\ 0 & & & G_N \end{bmatrix}, \end{aligned} \tag{18}$$

for suitable $m \times m$ matrices $G_1, \dots, G_N$. The matrix $\mathcal{G}$ can also be written as

$$\mathcal{G} = C(\boldsymbol{g})^N$$

where $\boldsymbol{g} = (G_N, G_{N-1}, \dots, G_1)$ and the block companion matrix $C$ associated with the block row vector $\boldsymbol{r} = (R_1, R_2, \dots, R_N)$ is defined as:

$$C(\boldsymbol{r}) = \begin{bmatrix} 0 & I & 0 & \dots & 0 \\ 0 & 0 & I & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 0 & I \\ R_1 & R_2 & \dots & \dots & R_N \end{bmatrix}. \tag{19}$$

# 3 The algorithms

First we discuss some general techniques and then specific algorithms for QBD and M/G/1 problems. For G/M/1 and non-skip-free problems we rely on the reduction to M/G/1 described in section 2.

## 3.1 Shift techniques

For the sake of simplicity, assume that the matrix equation (5) can be rewritten as

$$G = \sum_{i=-1}^{M} A_i G^{i+1}. \tag{20}$$

In practice, this is the rule for the decay properties of the blocks $A_i$. According to the sign of the drift $\mu$ define the following blocks $\widetilde{A}_i$, $i \geq -1$. If $\mu \leq 0$ set

$$\begin{aligned} &\widetilde{A}_{-1} = A_{-1}(I - Q), \\ &\widetilde{A}_i = A_i - (\textstyle\sum_{j=-1}^{i} A_j - I)Q, \quad 0 \leq i \leq M, \end{aligned}$$

where $Q = \boldsymbol{e}\boldsymbol{u}^T$ and $\boldsymbol{u}$ is any vector such that $\boldsymbol{e}^T\boldsymbol{u} = 1$. If $\mu > 0$ set

$$\begin{aligned}
&\widetilde{A}_{-1} = A_{-1} \\
&\widetilde{A}_0 = A_0 + EA_{-1} \\
&\widetilde{A}_i = A_i - E(I - \textstyle\sum_{j=-1}^{i-1} A_j), \quad 2 \le i \le M,
\end{aligned}$$

where $E = \boldsymbol{u}\boldsymbol{v}^T$, with $\boldsymbol{u}$ being any nonzero vector, and $\boldsymbol{v}$ such that $\boldsymbol{v}^T\boldsymbol{u} = 1$, $\boldsymbol{v}^T(\sum_{i=-1}^{M} A_i) = \boldsymbol{v}^T$.

Moreover, let us introduce the new equation

$$X = \sum_{i=-1}^{M} \widetilde{A}_i X^{i+1}. \tag{21}$$

It can be proved (see [4]) that the solution $\widetilde{G}$ of smallest spectral radius of (21) is

$$\begin{aligned}
&\widetilde{G} = G - Q \quad \text{if } \mu \le 0, \\
&\widetilde{G} = G \qquad\quad \text{if } \mu > 0.
\end{aligned}$$

For QBD problems the new equation turns into

$$X = \widetilde{A}_{-1} + \widetilde{A}_0 X + \widetilde{A}_1 X^2 \tag{22}$$

where for $\mu \le 0$ one has $\widetilde{A}_{-1} = A_{-1}(I-Q)$, $\widetilde{A}_0 = A_0 + A_1 Q$, $\widetilde{A}_1 = A_1$, whereas for $\mu > 0$ one has $\widetilde{A}_{-1} = A_{-1}$, $\widetilde{A}_0 = A_0 + EA_{-1}$, $\widetilde{A}_1 = (I-E)A_1$.

It has been proved that the roots of the polynomials $\widetilde{a}(z) = \det(\lambda I - \sum_{i=-1}^{M} z^{i+1}\widetilde{A}_i)$, and $a(z) = \det(\lambda I - \sum_{i=-1}^{M} z^{i+1} A_i)$, are the same except for the root $z = 1$ of $a(z)$ which is shifted to zero or to the infinity for $\widetilde{a}(z)$ according to the sign of the drift $\mu$.

This tiny difference on the roots of $a(z)$ and $\widetilde{a}(z)$ makes a great difference in the convergence speed of the algorithms for the solution of matrix equations if applied to (20) or to (21). In fact, iterative methods converge faster if applied to equation (21). For more details on this shift technique we refer the reader to [12], [4], [2].

## 3.2 Functional iterations

Methods based on functional iterations generate a sequence $\{X_k\}_k$ of matrices converging to the solution $G$ once $X_0$ has been suitably choosen. We recall the three main iterations called natural, traditional and U-based:

$$X_{k+1} = \sum_{i=-1}^{+\infty} A_i X_k^{i+1} \qquad \text{natural}, \tag{23}$$

$$X_{k+1} = (I - A_0)^{-1}\Big(A_{-1} + \sum_{i=1}^{+\infty} A_i X_k^{i+1}\Big) \qquad \text{traditional}, \tag{24}$$

$$X_{k+1} = \Big(I - \sum_{i=0}^{+\infty} A_i X_k^i\Big)^{-1} A_{-1} \qquad \text{U-based.} \tag{25}$$

If the initial matrix is $X_0 = 0$ then monotonic convergence of $X_k$ to $G$ occurs for all the three sequences. Convergence is linear if $\mu \neq 0$, sublinear if $\mu = 0$. If the initial matrix is $X_0 = \sigma I$, where $\sigma = \rho(G)$ and $\rho(\cdot)$ denotes the spectral radius, then monotonicity is lost but the convergence is faster though still linear. We recall that if $\mu \leq 0$ then $\sigma = 1$, if $\mu > 0$ then $\sigma$ is the smallest positive solution of the equation $z = \rho(A(z))$, where $A(z) = \sum_{i=-1}^{+\infty} z^{i+1} A_i$.

The three iterations above can be applied to the "shifted" equation (21). Local convergence is guaranteed but no analysis has been carried out concerning the convergence properties related to the choice of $X_0$. More details on functional iterations can be found in [4, 13, 18]. For non-skip-free Markov chains one has to compute the matrix $\mathcal{G}$ of (17). In this case it is sufficient to compute the blocks $G_1, \ldots, G_N$. These matrices can be approximated by means of the sequence of block row vectors $\boldsymbol{x}_k$, $k \geq 0$ generated from $\boldsymbol{x}_0$ by:

$$\boldsymbol{x}_{k+1} = [A_{-N}, A_{-N+1}, \ldots, A_{-1}] + \sum_{i=N}^{M+N} A_{i-N} \boldsymbol{x}_k C(\boldsymbol{x}_k)^{i-N}.$$

In fact, if $\boldsymbol{x}_0 = 0$ the sequence $\{\boldsymbol{x}_k\}$ monotonically converges to $[G_1, \ldots, G_N]$ of (18). The products $\boldsymbol{r}_j = \boldsymbol{x}_k C(\boldsymbol{x}_k)^j$ for $j = 0, \ldots, M$, are computed by means of the equation

$$\boldsymbol{r}_{j+1} = \boldsymbol{r}_j C(\boldsymbol{r}), \quad j = 0, 1, \ldots, M$$

starting with $\boldsymbol{r}_0 = \boldsymbol{x}_k$. This iteration has been introduced and analyzed in [10].

A faster convergence method is Newton's iteration [4], [14]. This method generates the sequence $X_{k+1} = X_k - W_n$, $k \geq 0$, $X_0 = 0$, where $W_k$ solves the linear matrix equation

$$W_k - K_k \sum_{i=1}^{+\infty} A_i \sum_{j=0}^{i-1} X_k^j W_k X_k^{i-j-1} K_k A_{-1} = X_k - K_k A_{-1}$$

and $K_k = (I - \sum_{i=0}^{+\infty} A_i X_k^i)^{-1}$. Its convergence is quadratic if $\mu \neq 0$. The above matrix equation can be solved by means of $O(m^6)$ arithmetic operations. For QBD Markov chains the complexity reduces to $O(m^3)$.

### 3.3 Invariant subspaces method

The *invariant subspaces* method consists in approximating the minimal nonnegative solution $G$ of the matrix equation (20) by approximating the left invariant subspace of a suitable block companion matrix. The method can be applied only if $\mu \neq 0$.

Define the matrix polynomial $H(t) = \sum_{i=0}^{N} t^i H_i$ as

$$H(t) = (1-t)^{N-1}(1+t)I - \sum_{i=0}^{N} (1-t)^{N-i}(1+t)^i A_{i-1},$$

and the matrices $\widehat{H}_i = H_N^{-1} H_i$, $i = 0, \ldots, N-1$. Set $\boldsymbol{h} = -[\widehat{H}_0, \ldots, \widehat{H}_{N-1}]$ and introduce the matrix

$$F = C(\boldsymbol{h}) + \mathrm{sign}(\mu)\boldsymbol{y}\boldsymbol{x}^T/(\boldsymbol{x}^T\boldsymbol{y}),$$

where

$$\boldsymbol{x}^{\mathrm{T}} = \left[\boldsymbol{x}_0^{\mathrm{T}}\widehat{H}_1 \;\; \boldsymbol{x}_0^{\mathrm{T}}\widehat{H}_2 \;\; \cdots \;\; \boldsymbol{x}_0^{\mathrm{T}}\widehat{H}_{N-1} \;\; \boldsymbol{x}_0^{\mathrm{T}}\right], \quad \boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

and $\boldsymbol{x}_0$, $\boldsymbol{y}_0$ are two vectors such that $\boldsymbol{x}_0^{\mathrm{T}}\widehat{H}_0 = 0$, $\widehat{H}_0\boldsymbol{y}_0 = 0$.

Define the matrix sign $S = \mathrm{Sign}(F)$ given by $S = \lim_k S_k$, where $S_k$ is the sequence generated by

$$\begin{cases} S_0 = F \\ S_{k+1} = \frac{1}{2}\left(S_k + S_k^{-1}\right), \quad k \geq 0. \end{cases} \tag{26}$$

Then $I - S$ has rank $m$ and by means of a rank revealing QR factorization of $I - S$ it is possibe to compute an $Nm \times m$ matrix $T$ whose columns are a basis of the linear space spanned by the columns of $I - S$. Denoting $T_1$ and $T_2$ the submatrices of $T$ made up by the rows $1, 2, \ldots, m$ and $m+1, m+2, \ldots, 2m$, it holds

$$G = (T_1 + T_2)(T_1 - T_2)^{-1}.$$

For positive recurrent and transient Markov chains the convergence of $S_k$ to $S$ is quadratic.

The matrix sign $S$ can be computed by means of the iteration (26) which can be stopped if $||S_k - S_{k+1}|| < \epsilon$ for some matrix norm $||\cdot||$ and for a given $\epsilon > 0$. A different approach is to compute $S$ by means of the Schur decomposition of $F$. Acceleration techniques have been devised based on the computation of some determinants. For more details and for the theory behind this technique we refer the reader to [4] and [1]. Comparisons between cyclic reduction and invariant subspace can be found in [19], [3].

## 3.4 Logarithmic reduction and cyclic reduction for QBD

Logarithmic reduction [15], [4] and cyclic reduction [5], [4] generate sequences of matrices converging quadratically to $G$ provided that $\mu \neq 0$. If $\mu = 0$, convergence generally turns to linear.

If these iterations are applied to the shifted equation (21) then quadratic convergence still occurs if $\mu = 0$.

## 3.5 Logarithmic reduction for QBD

Logarithmic reduction is synthesized by the following equations:

$$\begin{aligned} B_{-1}^{(k+1)} &= (C^{(k)})^{-1}(B_{-1}^{(k)})^2, \\ B_1^{(k+1)} &= (C^{(k)})^{-1}(B_1^{(k)})^2, \quad k \geq 0, \end{aligned} \tag{27}$$

where

$$C^{(k)} = I - B_{-1}^{(k)} B_1^{(k)} - B_1^{(k)} B_{-1}^{(k)} \tag{28}$$

and

$$B_{-1}^{(0)} = (I - A_0)^{-1} A_{-1}, \quad B_1^{(0)} = (I - A_0)^{-1} A_1.$$

The sequence

$$G_k = B_{-1}^{(0)} + \sum_{i=1}^{k} \left( \prod_{j=0}^{i-1} B_1^{(j)} \right) B_{-1}^{(i)} \tag{29}$$

converges to $G$.

Logarithmic reduction can be applied to the shifted equation (22) by defining $B_{-1}^{(0)} = (I - \widetilde{A}_0)^{-1} \widetilde{A}_{-1}$, $B_1^{(0)} = (I - \widetilde{A}_0)^{-1} \widetilde{A}_1$. The sequence $G_k$ of (29) converges to the solution $\widetilde{G}$ of (22).

## 3.6 Cyclic reduction for QBD

Cyclic reduction is synthesized by the following equations:

$$\begin{aligned}
K^{(k)} &= (I - A_0^{(k)})^{-1}, \\
A_{-1}^{(k+1)} &= A_{-1}^{(k)} K^{(k)} A_{-1}^{(k)}, \\
A_0^{(k+1)} &= A_0^{(k)} + A_{-1}^{(k)} K^{(k)} A_1^{(k)} + A_1^{(k)} K^{(k)} A_{-1}^{(k)}, \\
A_1^{(k+1)} &= A_1^{(k)} K^{(k)} A_1^{(k)}, \\
\widehat{A}_0^{(k+1)} &= \widehat{A}_0^{(k)} + A_1^{(k)} K^{(k)} A_{-1}^{(k)},
\end{aligned} \tag{30}$$

for $n = 0, 1, \ldots$, and $\widehat{A}_0^{(0)} = A_0$, $A_i^{(0)} = A_i$, $i = -1, 0, 1$. The sequence

$$G_k = (I - \widehat{A}_0^{(k)})^{-1} A_{-1} \tag{31}$$

converges to $G$.

Cyclic reduction can be applied to the shifted equation (22) by defining $\widehat{A}_0^{(0)} = \widetilde{A}_0$, $A_i^{(0)} = \widetilde{A}_i$, $i = -1, 0, 1$. If $\mu \leq 0$ the sequence $G_k$ of (31) converges to $G$. If $\mu > 0$ the sequence generated by $G_k = (I - \widehat{A}_0^{(k)})^{-1} \widetilde{A}_{-1}$ converges to $G$.

## 3.7 Cyclic reduction for M/G/1-type Markov chains

Cyclic reduction can be extended to M/G/1 Markov chains by means of a functional interpretation. Given a matrix power series $F(z) = \sum_{i=0}^{+\infty} z^i F_i$ define the matrix power series

$$\begin{aligned}
F_{\text{even}}(z) &= \frac{1}{2}(F(\sqrt{z}) + F(-\sqrt{z})) = \sum_{i=0}^{+\infty} z^i F_{2i}, \\
F_{\text{odd}}(z) &= \frac{1}{2\sqrt{z}}(F(\sqrt{z}) - F(-\sqrt{z})) = \sum_{i=0}^{+\infty} z^i F_{2i+1},
\end{aligned}$$

Let $A^{(0)}(z) = \sum_{i=-1}^{+\infty} z^{i+1} A_i$, $\widehat{A}^{(0)}(z) = \sum_{i=0}^{+\infty} z^i A_i$, and define

$$\begin{aligned} K^{(k)}(z) &= (I - A_{\text{odd}}^{(k)}(z))^{-1}, \\ A^{(k+1)}(z) &= z A_{\text{odd}}^{(k)}(z) + A_{\text{even}}^{(k)}(z) K^{(k)}(z) A_{\text{even}}^{(k)}(z), \\ \widehat{A}^{(k+1)}(z) &= \widehat{A}_{\text{even}}^{(k)}(z) + \widehat{A}_{\text{odd}}^{(k)}(z) K^{(k)}(z) A_{\text{even}}^{(k)}(z). \end{aligned} \tag{32}$$

Then the sequence

$$G_k = (I - \widehat{A}^{(k)}(0))^{-1} A_{-1} \tag{33}$$

converges to $G$. Convergence is quadratic if $\mu \neq 0$, and is generally linear if $\mu = 0$.

Cyclic reduction can be applied to the shifted equation (21) by defining $A^{(0)}(z) = \sum_{i=-1}^{+\infty} z^{i+1} \widetilde{A}_i$, $\widehat{A}^{(0)}(z) = \sum_{i=0}^{+\infty} z^i \widetilde{A}_i$. If $\mu \leq 0$ the sequence $G_k$ of (33) converges to $G$. If $\mu > 0$ the sequence generated by $G_k = (I - \widehat{A}^{(k)}(0))^{-1} \widetilde{A}_{-1}$ converges to $G$. If cyclic reduction is applied to the shifted equation (21) then convergence is quadratic even if $\mu = 0$.

Concerning the different ways for implementing cyclic reduction we refer the reader to the book [4, Chapter 7]. One of the most efficient implementations relies on the technique of evaluation/interpolation where the computation of the coefficients of the matrix power series $A^{(k+1)}(z)$ is performed by means of a point-wise computation of the right-hand side of (32) at the $q$th roots of the unity, followed by an interpolation stage. Here $q$ must be a sufficiently large integer such that $\sum_{i \geq q} ||A_i^{(k+1)}||$ is negligible for some matrix norm $||\cdot||$. Such an integer $q$ exists for the decay to zero of the coefficients $A_i^{(k+1)}$ and can be efficiently computed by means of a suitable technique of [6]. The method obtained in this way is called *point-wise cyclic reduction*, its efficiency relies on the use of FFT for performing the evaluation and the interpolation stages [6]. Its complexity is related to the number of interpolation points needed in the computation, or, equivalently, to the value of the numerical degrees of the matrix power series $A^{(k)}(z)$.

## 3.8 The Ramaswami reduction

The Ramaswami reduction [23], [4], [7] allows one to reduce an M/G/1-type Markov chain into a QBD process with blocks of infinite size. In this way, algorithms for solving a QBD can be adapted for solving an M/G/1-type Markov chain.

Define the matrices

$$\mathcal{A}_1 = \begin{bmatrix} 0 & & & 0 \\ I & 0 & & \\ & I & 0 & \\ & & \ddots & \ddots \\ 0 & & & \end{bmatrix}, \tag{34}$$

$$\mathcal{A}_0 = \begin{bmatrix} A_0 & A_1 & A_2 & \cdots \\ 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \tag{35}$$

and

$$\mathcal{A}_{-1} = \begin{bmatrix} A_{-1} & 0 & 0 & \cdots \\ 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{36}$$

Then the matrix

$$\mathcal{G} = \begin{bmatrix} G & 0 & 0 & \cdots \\ G^2 & 0 & 0 & \cdots \\ G^3 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{37}$$

is the minimal nonnegative solution of the equation $\mathcal{G} = \mathcal{A}_{-1} + \mathcal{A}_0\mathcal{G} + \mathcal{A}_1\mathcal{G}^2$, where $G$ is the minimal nonnegative solution of (2).

Logarithmic reduction and cyclic reduction can be applied for computing $\mathcal{G}$ in order to compute $G$. The shift technique of section 3.1 can be applied either to the original equation or to the infinite QBD obtained after the Ramaswami reduction.

# 4 Fortran 95 subroutines

Most part of the algorithms described in previous sections has been implemented in Fortran 95. The subroutines can be called by the user inside a main program, using an appropriate module, and linking then the program with the libraries libsmcsolver.a, lapack95, lapack, blas (see Section 6 for more detail on compilation).

Appendix A list all the symbols of functions, subroutines, and variables defined in the library libsmcsolver.a. Most of them are used internally. We document here the subroutines and the variables that are of interest for a user who writes his own main program and wishes to use the algorithms implemented in the library.

## 4.1 Basic Routines

The first two subroutines `crqbd` and `lrqbd` compute the minimal solutions $G$, $R$ and $U$ of the equations (2). They have the same sintax, i.e.,

```
   use smc_int
subroutine crqbd(An1, A0, A1, G, R, U, doshift, dogth,&
     drift, nerror, maxit)
subroutine lrqbd(An1, A0, A1, G, R, U, doshift, dogth,&
     drift, nerror, maxit)
```

- `An1, A0` and `A1` are the matrix variables containing the matrices $A_{-1}$, $A_0$ and $A_1$ respectively;
- The output variable `G` contain the solution $G$;
- The optional output variables `R` and `U` contain the solutions $R$ and $U$, respectively;
- the logical variables `doshift` and `dogth`, if `.true.`, perform the shift acceleration of [8, Section III] and the GTH trick of [11] for improving numerical stability. This trick cannot be applied if the shift acceleration is performed;
- `drift` contains the value of the drift $\mu$;
- The optional input variables `nerror` and `maxit` contain the error bound for the stop condition in the form of an integer exponent (the actual error is $epsilon(1.d0) * 10.0d0 ** nerror$, default $nerror = 0$ ) and the maximum number of allowed iterations (by default `maxit=50`), respectively.

The subroutines `pwcr` and `spwcr` compute the solution of the equation (5), (or to be more precise of (6)) by means of the algorithm of point-wise cyclic reduction without the shift acceleration (`pwcr`) and with the shift acceleration (`spwcr`). Their sintax is the same, i.e.,

```
   use pwcr_interface
subroutine pwcr(A, eps, G, err, maxit, intpmax)
subroutine spwcr(A, eps, G, err, maxit, intpmax)
```

- `A` is a three-way array where `A(:,:,k)` contains the block $A_{k-2}$ for $k = 1, 2, \ldots, K+2$, (where $K$ is such that $\sum_{i=-1}^{K} A_i$ is numerically stochastic);
- `eps` contain an error bound used in the stop condition (the default value is `eps=1e-12`);
- The output variable `G`, contains the solution $G$ of the matrix equation (6);
- `err` is the residual error;
- `maxit` and `intpmax` are the maximum number of iterations (by default `maxit=50`) and an index of the maximum number of interpolation points ($= 256 * 2^{intpmax}$) (by default `intpmax=1` and the number of interpolation points is 256), respectively.

The subroutine `fi` computes the solution $G$ of (2) or (6) by means of functional iterations. The syntax is

```
   use fi_int
subroutine fi(A, doshift, method, eps, maxit,&
     x0, G, drift, err)
```

- `A` is a three-way array: for QDB problems `A(:,:,k)` contains the blocks $A_{-1}, A_0, A_1$, for M/G/1 problems contains the blocks $A_{-1}, A_0, \ldots, A_K$, (where $K$ is such that $\sum_{i=-1}^{K} A_i$ is numerically stochastic) ;
- `doshift` is an optional logical variable, if `.true.` it performs the shift technique (default `doshift=.false.`);

- `method` is an optional integer variable which selects the functional iteration: 1→Natural iteration, 2→Traditional iteration, 3→Method based on the matrix $U$ (default `method=3`);
- optional input variables `eps` and `maxit` contain an error bound on the stop condition (by default `eps=1e-12`) and the maximum number of iterations (by default `maxit=10000`);
- `X0` is an optional input variable which contains the initial approximation (by default `X0=0` if `doshift=.true.` and `X0=rI` if `doshift=.false.` with `r`=$\rho(G)$);
- output variable `G` contains the solution $G$ of the matrix equation (2) or (6)
- output variables `drift` and `err` contain the drift $\mu$ and the residual error, respectively.

The subroutine `is` implements the invariant subspace method for solving (2) or (6). Its sintax is

```
   use is_int
subroutine is(A, method, eps, maxit, G, drift, err)
```

- `A` is a three-way array: for QDB problems `A(:,:,k)` contains the blocks $A_{-1}, A_0, A_1$, for M/G/1 problems contains the blocks $A_{-1}, A_0, \ldots, A_K$, (where $K$ is such that $\sum_{i=-1}^{K} A_i$ is numerically stochastic);
- `method` is an optional integer variable which selects the algorithm for computing the invariant subspace: 1→ Matrix Sign Iteration; 2→ Matrix Sign Iteration with the Balzer acceleration; 3→ Schur Decomposition; by default `method=2`;
- The optional input variables `eps` and `maxit` contain an error bound on the stop condition (by default `eps=1e-12`) and the maximum number of iterations (by default `maxit=50`);
- The output variable `G` contain the solution $G$ of the matrix equation (2) or (5);
- The output variables `drift` and `err` contain the drift $\mu$ and the residual error, respectively.

## 4.2 Auxiliary Routines

The following routine converts a G/M/1 problem in an M/G/1 problem.

Its sintax is

```
   use smc_int
subroutine gm1tomg1(A, AA, v)
```

- `A` is the input three-way array where `A(:,:,k)` contains the block $A_{2-k}$ for $k = 1, 2, \ldots, K+2$ of the original G/M/1 problem.

- `AA` is the output three-way array where `AA(:,:,k)` contains the block $A_{k-2}$ for $k = 1, 2, \ldots, K+2$ of the equivalent M/G/1 problem;
- `v` is a vector containing the strictly positive stationary probability vector of $A$ that is computed during the conversion.

The following routines compute $U$ from $G$ and $R$ from $U$, respectively, in QBD problems. Their sintax is

```
   use smc_int
subroutine gtou(A0, A1, G, U)
subroutine utor(A1, U, R)
```

- `A0` and `A1` are the matrix variables containing the matrices $A_0$ and $A_1$ respectively;
- The variables `G R` and `U` contain the solutions $G$ $R$ and $U$, respectively;

The following routines compute the norm of the differences between left and right side of equations (2), (6), (11) for the numerical solutions. Their sintax is

```
   use smc_int
subroutine qbdrres(Am1, A0, A1, R, res)
subroutine qbdures(Am1, A0, A1, U, res)
subroutine rresidual(A, R, res)
   use fi_int
subroutine gresidual(A,G,res)
```

- `Am1, A0` and `A1` are the matrix variables containing the matrices $A_{-1}$, $A_0$ and $A_1$ respectively, in the QBD problem;
- `A` is a three-way array: for M/G/1 problems `A(:,:,k)` contains the blocks $A_{k-2}$, for $k = 1, 2, \ldots, K+2$ for G/M/1 problems `A(:,:,k)` contains the blocks $A_{2-k}$ for $k = 1, 2, \ldots, K+2$;
- The variables `G` and `R` contain the numerical solutions $G$ and $R$ respectively;
- `res` is the computed norm.

To compute $\boldsymbol{\pi}$ we have the following routines:

```
   use pi_int
subroutine qbdpi(Am1, A0, A1, Bm1, B0, B1, R, &
     maxnc, epspi, pi0, pi, pinc)
subroutine mg1pi(A, B0, B, Bm1, G, &
     maxnc, epspi, pi0, pi, pinc)
subroutine gm1pi(A, B, B0, Bm1, R, &
     maxnc, epspi, pi0, pi, pinc)
```

- for QBD problems `Am1, A0` and `A1` are the matrix variables containing the matrices $A_{-1}$, $A_0$ and $A_1$ respectively;

- `A` is a three-way array: for M/G/1 problems `A(:,:,k)` contains the blocks $A_{k-2}$, for $k = 1, 2, \ldots, K+2$
- `A` is a three-way array: for G/M/1 problems `A(:,:,k)` contains the blocks $A_{2-k}$ for $k = 1, 2, \ldots, K+2$;
- for QBD problems `Bm1, B0` and `B1` are the matrix variables containing the matrices $B_{-1}$, $B_0$ and $B_1$ respectively;
- `B0` is the matrix $B_0$;
- `B1` for QBD and M/G/1 is the matrix $B_1$;
- `Bm1` for QBD and M/G/1 is the matrix $B_{-1}$; for G/M/1 is the matrix $B_1$
- `B` is a three-way array: for M/G/1 problems `B(:,:,k)` contains the blocks $B_{2-k}$, for $k = 1, 2, \ldots, K+2$, for G/M/1 problems it contains the blocks $B_{k-2}$ for $k = 1, 2, \ldots, K+2$;
- `G, R` contain the solutions $G$ and $R$ respectively;
- `maxnc` contains the maximum number of vectors $\boldsymbol{\pi}_k$ to compute.
- `epspi` contain an error bound used in the stop condition (the default value is `eps=1e-12`);
- `pi0, pi` are the $\boldsymbol{\pi}_0$ and a vector containig the $\boldsymbol{\pi}_k$ of (3) for QBD, of (7) and (9) for M/G/1, and of (12) and (13) for G/M/1 problems, respectively;
- `pinc` is the number of computed blocks of components of $\boldsymbol{\pi}$.

## 4.3 Internal Routines

`function binomial(n,k)`

- `n`, `k` are integers
- the output of the function is a double precision real.

## 4.4 Blas and Lapack Routines

There are very few calls from the SMCSOLVER routines to Blas, Lapack and Lapack95 routines:

`la_gesv`
`la_getrf`
`la_getri`
`la_gees`
`dgeqp3`
`dorgqr`
These, in turn, call some other Blas and Lapack routines, and so on: as a rule, names of Blas, Lapack and Lapack95 routines cannot be reassigned.

## 4.5 Global Variables

Variables in Module `ponte_f_f`:

- `fdrift`
  some routines save the drift value into this variable; this has been a last minute emergency patch and will be changed in future versions.
- `debug`
  if `debug=.true.` some debugging printouts will appear in some situations; it is not a sistematic feature, it did help in debugging the code; anyway the user can add some code lines like
  if (debug) write(*,*) "some message or some data..."
  anywhere in the code to check for some values, and switch the printouts on or off just changing the value of the variable `debug`.
- `verb`
  this variable controls the verbosity of the printouts during execution of code; if `verb=.true.` at each iteration there will be a printout otherwise just a `*` will be printed.

Variables in Module `smc_tools`:

- `dp`
  dp=kind(0.d0) is used in dimensionig real double precision, as a shortcut.
- `info`
  usually 0, is different from 0 if some allocation has failed.

# 5 The graphical interface

We have implemented a simple *Graphical Interface* in C relying on the **GTK+ 2.0** graphical libraries.

The executable runs in Linux Workstations where the corresponding GTK shared libraries are available.

The Fortran routines are usually linked statically, so that the executable needs neither Fortran Compiler nor Fortran run-time libraries to run.

A fully statically linked program is discouraged by GTK+ developers.

## 5.1 The menus

By means of menus in the main window (Figure 1), the user can easily choose the kind of problem (QBD, M/G/1, G/M/1), either by loading the input data from some external ASCII files or by loading one of the examples provided in the package.

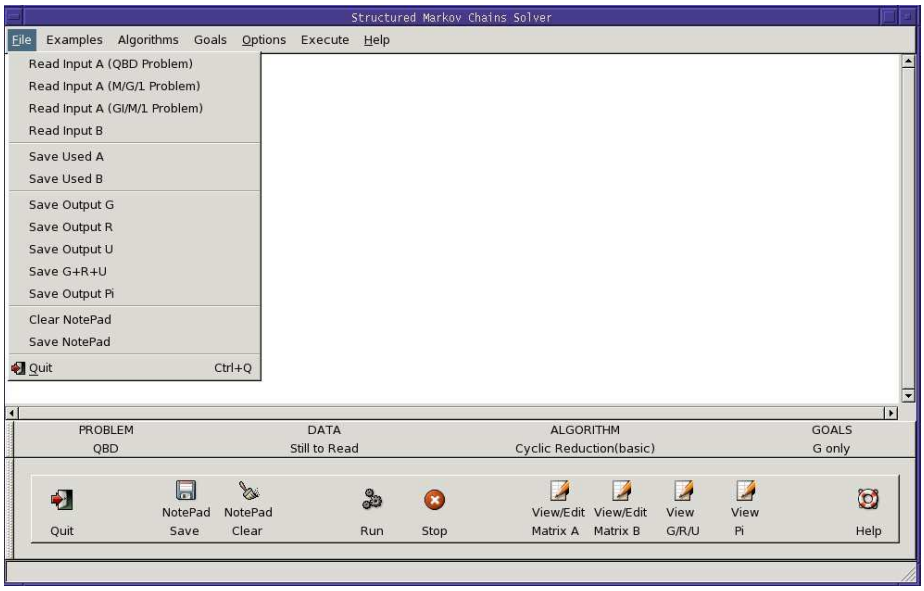


Figure 1: Graphical User Interface: main window, file menu

Several formats are supported for the input files. In a first step the file is opened, read and analyzed in an attempt to discover its format; in a second step the matrices are allocated, the file is re-read and the matrices are loaded in memory. Fortran formats need some simple header with dimensioning data. Matlab formats rely on counting number of lines and number of floats in a line. More detail in the next section.

Selecting one of the examples from the Examples menu, an example window (Figure 2) will appear, allowing the user to to set up some parameters for the selected Example.

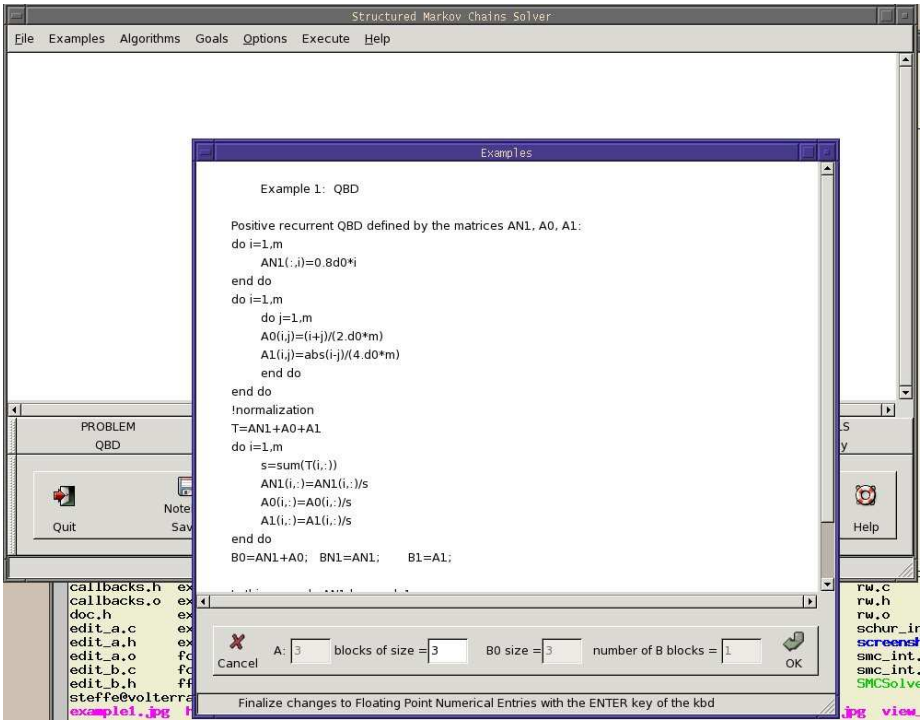


Figure 2: Graphical User Interface: examples window

From the main window's menu Algorithms (Figure 3) the user can choose an algorithm among Cyclic Reduction, Logarithmic Reduction, Functional Iterations, Invariant Subspace. For each algorithm the user can set additional optional parameters like the shift acceleration, the diagonal adjustment, or can modify the values of the maximum number of iterations (default 50, range from 5 to 15000), the number of interpolations points in cyclic reduction (defaut 512, range from 256 to 65536), the error bound for the stop condition (default $10^{-12}$, range from $10^{-5}$ to $10^{-16}$). Morover, in the case of the Functional Iterations Algorithm, the starting matrix can be read from external file (internal choices are the null and the identity matrices).

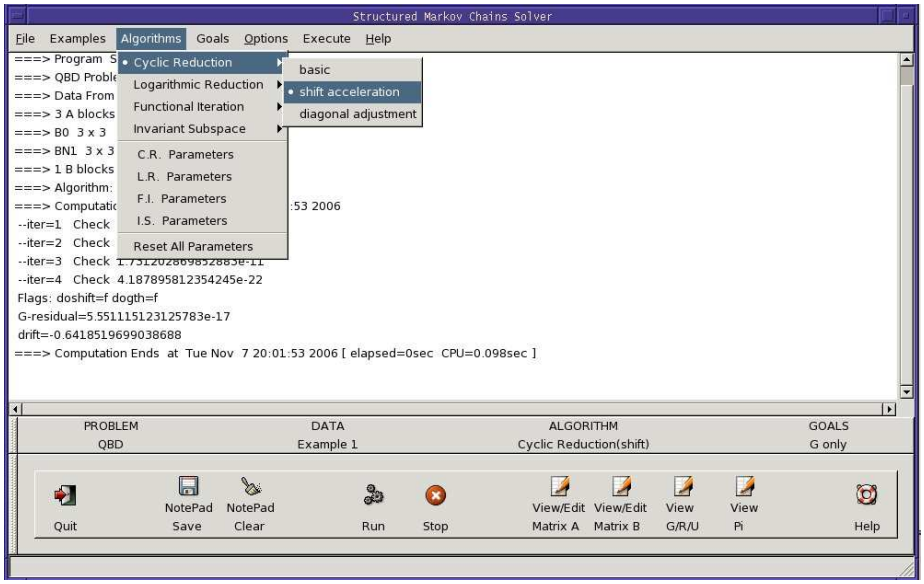


Figure 3: Graphical User Interface: main window, algorithms menu

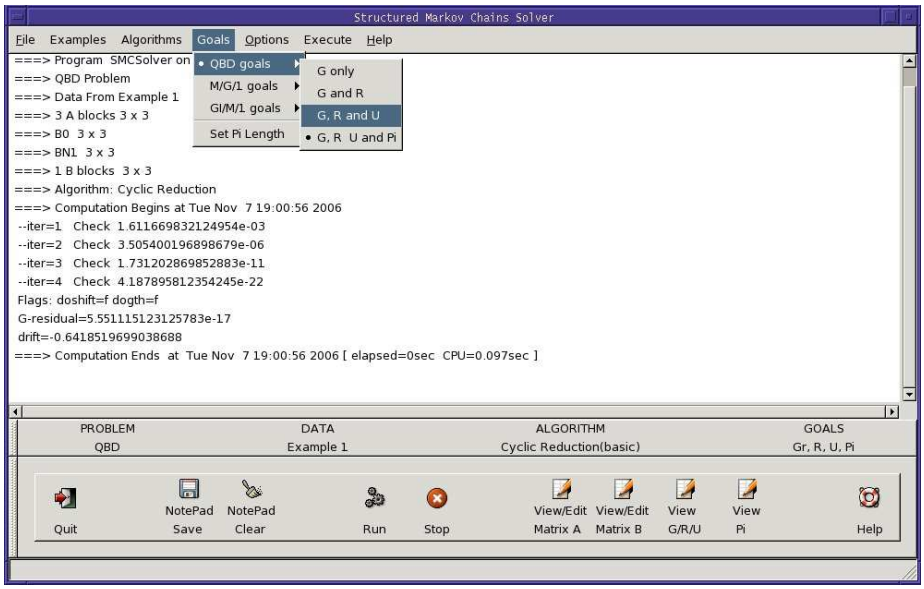


Figure 4: Graphical User Interface: main window, goals menu

For each kind of problem, the user can select the goals (Figure 4), for instance, computing $G$, $R$, $U$ or $\boldsymbol{\pi}$. The length of Pi is selectable in the same menu.

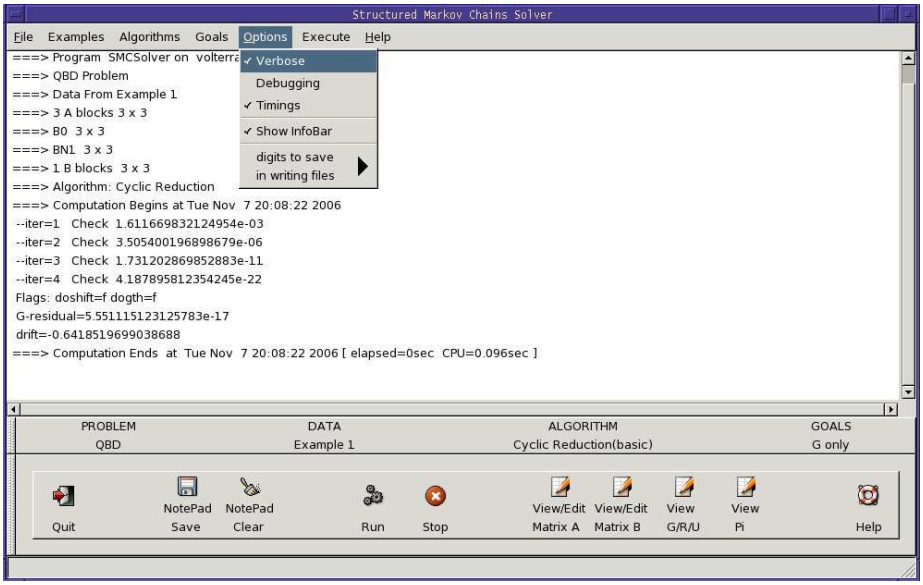


Figure 5: Graphical User Interface: main window, options menu

Warnings are flashed in the status line of the window for inconsistent choices of the user and also if the selected algorithms or options are not available.

The Options menu allows to select the level of verbosity and the presence of timing information. The debug option will print on the tty some messages of interest for the developers only. The number of digits to write in saving ASCII outputs files can be set up in Options Menu too.

The use of multithread support enables the user to start/stop a computation just by pressing a button.

During the numerical computation, partial results, timings, and information on the evolution of the computation are displayed in real time on a scrollable viewport of the main window. The buffer of this viewport with the log of the session can be appended to a file.

A simple online window help is available.

## 5.2 Matrix Editor

A viewing and editing graphical tool allows to display and edit input matrices and to view and compare output matrices. This is performed in a graphical way by representing each element of the matrices as a small gray or colored square, where the intensity varies according to the magnitude of the number.

A good choice of the color scale allows an immediate and intuitive viewing of the structure of the matrix:

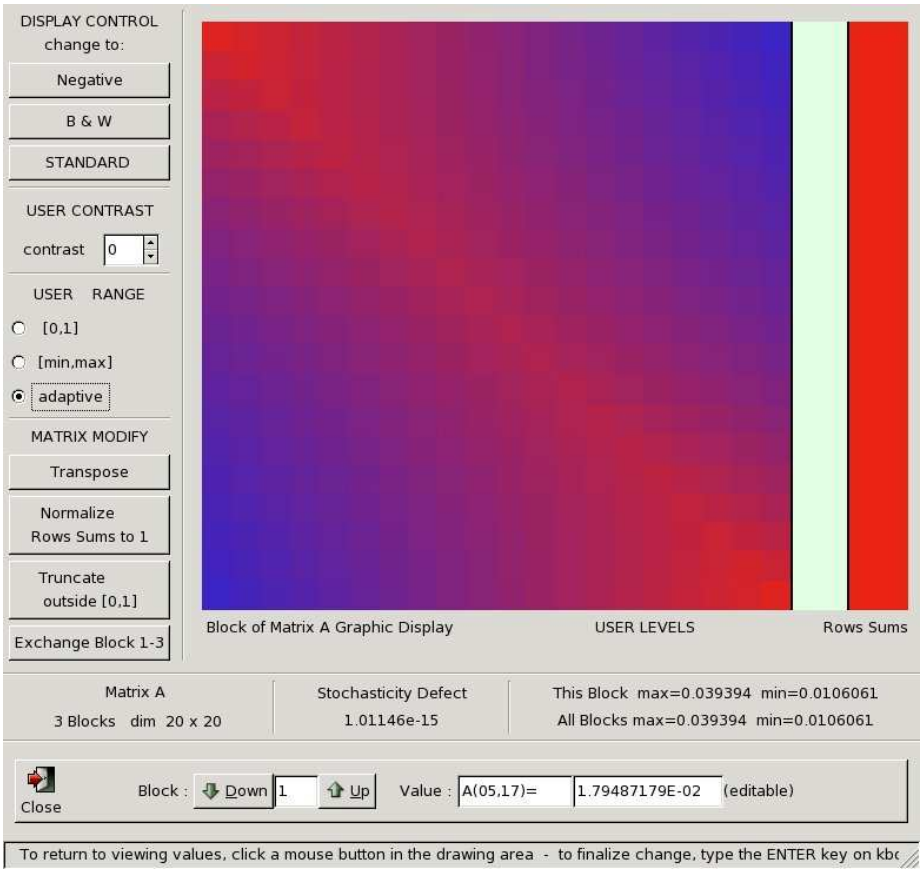


Figure 6: Graphical User Interface: Edit A

The standard color scale is a linear one. User color scale can be linear in the range $[0, 1]$ or in the range between the minimun nonzero value and the maximum value; adaptive scale is a non linear color scale, that attemps to assign as many of the 256 levels as possible.

The numerical values of the entries are displayed and edited in a separate small window as the user moves the cursor on the colored squares of the matrix area.

To edit a value, click a mouse botton: the entry with the matrix indexes will freeze, and the entry with the value becomes editable; change the value and finalize the change with the Enter key.

In View/Edit Matrix A the block index varies from 1 to $K + 2$; these values for QBD problems corresponds to $A_{-1}, A_0, A_1$, for M/G/1 corresponds to $A_{-1}, A_0, A_1, \ldots, A_K$, and for G/M/1 corresponds to $A_1, A_0, A_{-1}, \ldots, A_{-K}$.

Matrices $G$, $R$, $U$, or the vector $\boldsymbol{\pi}$ can be viewed but not edited. It is possible to compare them with matrices present in a file: comparison will compute the

norm of the difference between the matrix present in memory and the matrix read from the target file.

## 5.3 File Formats

The computed solutions (matrices, vectors) can be saved in several ASCII formats. Morover a sparse matrix format is suppported for reading or writing data (computation do not use sparse matrix format internally).

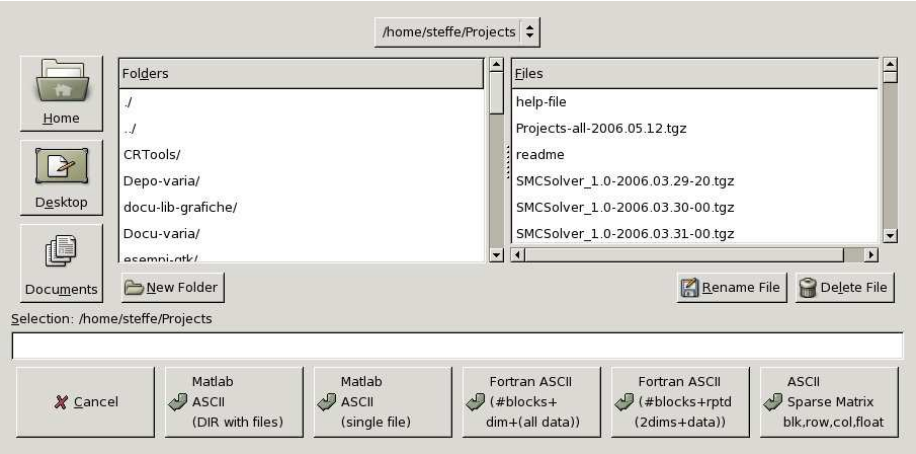


Figure 7: Graphical User Interface: saving $A$

Remember that the number of digits that are written for every number saved in ASCII Files is selectable in the Options Menu (Figure 5). The choices are 8,15,16,17 digits in the mantissa; the default is 16 digits.

We list all the available Save File Formats:

`Save G or U or R`:

- `Matlab`
  the floats of a matrix row are written in one line, Matlab counts the number of lines and the number of floats in each line for detecting the number of rows and colums of the matrix;
- `Fortran ASCII 2dim+data`
  the first line holds two integers, the number of rows and of columns of the matrix; the second (very long) line holds all the floats, column after column, with a space between floats and at the beginning of each line;
- `Fortran ASCII 1dim+data`
  the matrix is a square one, and the first line holds one integer, the number of rows equal to the number of colums; the second (very long) line holds

all the floats, column after column, with a space between floats and at the beginning of each line, as before.

`Save G,U,R` (QBD only):

- `Matlab dir with 3 files`
  a directory is created and the 3 matrices $G$, $R$, $U$ are saved as Matlab ASCII matrices in the files `G, R, U`.
- `Fortran 3 x (2 dim + data)`
  the 3 matrices $G$, $R$, $U$ are written in one file, one after the other, each in two lines, first line holds two integers, the number of rows and of columns of the matrix; the second (very long) line holds all the floats, column after column, with a space between floats and at the beginning of each line;
- `Fortran 1 dim + 3 x data`
  the first line holds one integer, the number of rows or of columns (they are equal), the second line holds all the floats of the 3 matrices $G$, $R$, $U$, one after the other, column after column, with a space between floats and at the beginning of each line.

`Save Pi`:

- `Matlab`
  $\boldsymbol{\pi}$ is written in Matlab ASCII style, as a single line;
- `Fortran dim+data`
  The file has two lines: the first line hold one integer, the number of components of $\boldsymbol{\pi}$, and the second line holds the floats, with a space between floats and at the beginning of each line.

`Save A`:

- `Matlab dir with files`
  a directory is created; each of the blocks $A(:,:,k)$, with $k = 1, 2, \ldots, K+2$, is saved in a separate file, in Matlab ASCII style and with names `A00001, A00002, ....`
- `Matlab single file`
  the blocks $A(:,:,k)$, with $k = 1, 2, \ldots, K+2$, are put side by side in one row; the resulting rectangular matrix is written in the file, in Matlab ASCII style. The number of columns divided the number of rows is the number of (square) blocks.
- `Fortran #blocks , + repeated (dims, data)`
  the first line holds one integer, the number of blocks; for each block there follows two lines, the first with two integers, row and columns number, the second (very long) line holds all the floats of the block, column after column, with a space between floats and at the beginning of each line;
- `Fortran #blocks, dims, + all data`
  the first line holds two integers, the number of blocks and the number of row (that is equal to the number of columns); the second (very long) line holds all the floats, block after block, and in each block column after column, with a space between floats and at the beginning of each line;

- `Fortran Sparse Matrix format`
  the first line has 3 integers, the number of blocks, of rows, and of columns; there follows one line for each non zero term of the matrix, holding 3 integers and one float, i.e. block index $k$, row index $i$, column index $j$, and the float $A(i, j, k)$.

`Save B`:

- `B0 Matlab`
  the floats of a matrix row are written in one line, Matlab counts the number of lines and the number of floats in each line for detecting the number of rows and colums of the matrix;
- `BN1 Matlab`
  the floats of a matrix row are written in one line, Matlab counts the number of lines and the number of floats in each line for detecting the number of rows and colums of the matrix;
- `B Matlab`
  the blocks $B(:, :, k)$, with $k = 1, 2, \ldots, K + 2$, are put side by side in one row; the resulting rectangular matrix is written in the file, in Matlab ASCII style. The number of columns is the product of the number of blocks by the number of columns of each block. One of the two numbers must be known, otherwise reconstructing the three way matrix $B$ from the file will be impossible.
- `B0 Fortran`
  the first line holds two integer, rows and columns numbers; the second line holds all the floats, column after column, with a space between floats and at the beginning of each line;
- `BN1 Fortran`
  the first line holds two integer, rows and columns numbers; the second line holds all the floats, column after column, with a space between floats and at the beginning of each line;
- `B Fortran`
  the first line holds three integers, the number of blocks, the number of rows and the number of columns of the three way matrix $B$; for each block there is one line holding all the floats, column after column, with a space between floats and at the beginning of each line;

`Read Files`:

The program will do its best to understand the format of the input file and to read the data. It will detect the presence of ASCII chars $> 127$, and invalidate the file. It will discard lines beginning with # or ! or % or @ considering them comment text. It will try to detect Matlab ASCII files, as files than consists of consecutive lines with the same number of floats and nothing else (no comments or extra chars allowed in Matlab files !). Fortran data files are

supposed to have some form of header with integers referring to dimensions of the matrices. Sparse format files will have lines with 3 indexes and a float.

Be aware that the routine that analize the floats will not read correctly very long integers ($> 9$ digits) or very long float (mantissa $> 17$ digits or exponent $> 3$ digits).

# 6 Compiling SMCSolver_1.2

To compile the program you will need:

- A Linux Workstation, complete with usual development programs;
- GNU gcc, make;
- libraries and header files of gtk+-2, gthread-2.0, ncurses and librt;
- A Fortran 95 Compiler like Lahey/Fujitsu (R) lf95, of Intel (R) Fortran Compiler ifort or f95 NAG(R) Fortran Compiler or GNU g95;
- the SMCSolver_1.2 sources (from http://bezout.dm.unipi.it/SMCSolver/ or from one of the authors by e-mail).

Untar the sources, and change the `make.inc`, uncommenting the lines for your Fortran Compiler and commenting the lines for different Fortran Compilers. In case you mess up the file, `make.inc.default` has the original version. The files `make.inc.g95`, `make.inc.ifort`, `make.inc.lf95`, `make.inc.nag` are samples tested to work respectively with GNU g95, Intel Fortran, Lahey Fortran, NAG Fortran.

If you have installed on your Workstation optimized versions of BLAS, LAPACK, LAPACK95 libraries and modules, let the variable FFLIBS point to the directory containing them. Otherwise you can use the minimum set of routines extracted from BLAS, LAPACK, LAPACK95 and included in the distribution. In this case the compiler first step will be to compile the required libraries and modules.

Remember that usually the linker requires that a library has a name beginning with lib (like libblas.a or liblapack.a ) and refers to it with the flags -I (for modules directory), -L (for library directory), -l ( for libraries, like -lblas -llapack).

When you are ready to compile, you can `make` the program:

- `make` will compile the program SMCSolver;
- `make clean` will delete modules, objects and the executable;
- `make cleanall` will delete libraries, modules, objects and the executable;
- `make test-lib` will compile the library `libsmcsolver.a` and a sample main program `test-lib.f90`, linking it to the library; In this case C compiler and graphic libraries are not needed.
- `make dist` must NOT be run by users, as it will change the `TIMESTAMP` and recreate a `SMCSolver_1.2.ACTUAL_DATE.tgz`.

- several other less interesting targets are availiable for the make command.

Several warnings from the compilers are normal. Errors in compilation can arise from several situations:

- if BLAS, LAPACK, LAPACK95 libraries and modules are not present in the system, the minimum set of them should be created by Fortran Compiler as first step. These routines are well known and well debugged: an error at this stage could mean a wrong flag passed to Fortran compiler in the file `make.inc`.
- gcc will then compile the 11 C files, without linking them. An error at this stage may point to missing or misplaced headers or non standard gcc on your Linux installation.
- then the Fortran Compiler will compile the 15 Fortran 90 files. It may fail if the Fortran Compiler lines in the `make.inc` are not correct for your compiler.
- last step, the Fortran Compiler has to link all the objects with both static and run time libraries, and this may fail if some library is not available, or if some option in `make.inc` is not correct for your Fortran Compiler.
- if you change the Fortran Compiler, remember to do a `make cleanall` as the Fortran libraries and modules of one Fortran Compiler will NOT work with a different Compiler.

SMCSolver will need several shared run time libraries to execute and will not start if some of these are missing. Moreover several icons and graphical details of the windows are loaded dinamically from files of the running host by gtk+-2.

If you are in doubt about the installed version of gtk libraries, use the command

```
pkg-config --modversion gtk+-2.0
```

Any version 2.x.y should be ok.

Compiling the library `libsmcsolver.a` does not requires the graphical libraries. To use a subroutine of the library, a Fortran 95 main program must declare the use of the appropriate modules, and must be linked with the library libsmcsolver.a, with lapack95, lapack and blas.

A simple example is the file `test-lib.f90`.

# Acknowledgment

This work has been partially supported by MIUR grant number 2004015437.

# References


[1] N. Akar and K. Sohraby. An invariant subspace approach in $M/G/1$ and $G/M/1$ type Markov chains. *Comm. Statist. Stochastic Models*, 13(3):381–416, 1997.

[2] D. A. Bini, L. Gemignani, and B. Meini. Solving certain matrix equations by means of Toeplitz computations: algorithms and applications. In *Fast algorithms for structured matrices: theory and applications (South Hadley, MA, 2001)*, volume 323 of *Contemp. Math.*, pages 151–167. Amer. Math. Soc., Providence, RI, 2003.

[3] D. A. Bini, G. Latouche, and B. Meini. Solving matrix polynomial equations arising in queueing problems. *Linear Algebra Appl.*, 340:225–244, 2002.

[4] D. A. Bini, G. Latouche, and B. Meini. *Numerical methods for structured Markov chains*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 2005. Oxford Science Publications.

[5] D. A. Bini and B. Meini. On the solution of a nonlinear matrix equation arising in queueing problems. *SIAM J. Matrix Anal. Appl.*, 17(4):906–926, 1996.

[6] D. A. Bini and B. Meini. Improved cyclic reduction for solving queueing problems. *Numer. Algorithms*, 15(1):57–74, 1997.

[7] D. A. Bini, B. Meini, and V. Ramaswami. Analyzing M/G/1 paradigms through QBDs: the role of the block structure in computing the matrix $G$. In G. Latouche and P. Taylor, editors, *Advances in Algorithmic Methods for Stochastic Models*, pages 73–86. Notable Publications, New Jersey, USA, 2000. Proceedings of the Third Conference on Matrix Analytic Methods.

[8] D. A. Bini, B. Meini, S. Steffé, and B. Van Houdt. Structured Markov chains solver: algorithms. *Proceedings of SMCTOOLS, Pisa 2006*, pages –, 2006.

[9] D. A. Bini, B. Meini, S. Steffé, and B. Van Houdt. Structured Markov chains solver: software tools. *Proceedings of SMCTOOLS, Pisa 2006*, pages –, 2006.

[10] H. R. Gail, S. L. Hantler, and B. A. Taylor. Non-skip-free $M/G/1$ and $G/M/1$ type Markov chains. *Adv. in Appl. Probab.*, 29(3):733–758, 1997.

[11] W. K. Grassmann, M. I. Taksar, and D. P. Heyman. Regenerative analysis and steady state distributions for Markov chains. *Oper. Res.*, 33(5):1107–1116, 1985.

[12] C. He, B. Meini, and N. H. Rhee. A shifted cyclic reduction algorithm for quasi-birth-death problems. *SIAM J. Matrix Anal. Appl.*, 23(3):673–691 (electronic), 2001/02.

[13] G. Latouche. Algorithms for infinite Markov chains with repeating columns. In *Linear algebra, Markov chains, and queueing models (Minneapolis, MN, 1992)*, volume 48 of *IMA Vol. Math. Appl.*, pages 231–265. Springer, New York, 1993.

[14] G. Latouche. Newton's iteration for non-linear equations in Markov chains. *IMA J. Numer. Anal.*, 14(4):583–598, 1994.

[15] G. Latouche and V. Ramaswami. A logarithmic reduction algorithm for quasi-birth-death processes. *J. Appl. Probab.*, 30(3):650–674, 1993.

[16] G. Latouche and V. Ramaswami. *Introduction to matrix analytic methods in stochastic modeling*. ASA-SIAM Series on Statistics and Applied Probability. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999.

[17] B. Meini. An improved FFT-based version of Ramaswami's formula. *Stochastic Models*, 13(2):223–238, 1997.

[18] B. Meini. New convergence results on functional iteration techniques for the numerical solution of $M/G/1$ type Markov chains. *Numer. Math.*, 78(1):39–58, 1997.

[19] B. Meini. Solving QBD problems: the cyclic reduction algorithm versus the invariant subspace method. *Adv. Perf. Anal.*, 1:215–225, 1998.

[20] M. F. Neuts. *Matrix-geometric solutions in stochastic models*, volume 2 of *Johns Hopkins Series in the Mathematical Sciences*. Johns Hopkins University Press, Baltimore, Md., 1981. An algorithmic approach.

[21] M. F. Neuts. *Structured stochastic matrices of $M/G/1$ type and their applications*, volume 5 of *Probability: Pure and Applied*. Marcel Dekker Inc., New York, 1989.

[22] V. Ramaswami. A stable recursion for the steady state vector in Markov chains of $M/G/1$ type. *Comm. Statist. Stochastic Models*, 4(1):183–188, 1988.

[23] V. Ramaswami. The generality of Quasi Birth-and-Death processes. In A. S. Alfa and S. R. Chakravarthy, editors, *Advances in Matrix Analytic Methods for Stochastic Models*, pages 93–113. Notable Publications, NJ, 1998.

## APPENDIX A

We list here the content of all the Fortran 95 modules:

module `fft_interface.mod`

subroutine iffts1
subroutine ffts2
subroutine ffts1
subroutine fillroots
subroutine fft1
subroutine ifft1
subroutine ftb1
subroutine ftb2
subroutine iftb2
subroutine iftb1
subroutine twiddle
subroutine itwiddle

module `fi_int.mod`

subroutine fi
subroutine fi_n
subroutine fi_t
subroutine fi_u
subroutine gresidual

module `is_int.mod`

subroutine is
function drft_mg1

module `pi_int.mod`

subroutine qbdpi
subroutine mg1pi
subroutine gm1pi

module `ponte_f_f.mod`

A
B
G
R
U
Pi
X0
B0
BN1
Pi0
fdrift
debug
verb
timing
finish
wout

module `pwcr_interface.mod`

subroutine pwcr
subroutine computeG
subroutine residual
subroutine spwcr
subroutine scomputeG

module `roots.mod`

wr
wi
wwr
wwi

module `schur_interface.mod`

subroutine schur
subroutine test
subroutine sc1p
subroutine sschur
subroutine stest
subroutine sc2p
subroutine scc2p
subroutine sc1
subroutine sc2
subroutine prodc
subroutine means
subroutine pmeans

subroutine scc2
subroutine solver
subroutine solvec

module `smc_int.mod`

subroutine drft
subroutine gth
subroutine crqbd
subroutine lrqbd
subroutine bgth
subroutine shift
subroutine gm1tomg1
subroutine rresidual
subroutine gtou
subroutine utor
subroutine qbdrres
subroutine qbdures

module `smc_tools.mod`

dp
info

Other Routines:

function binomial
function sel
subroutine append_fifo
subroutine print_it
subroutine print_it_
subroutine print_it_nolf
subroutine print_it_nolf_