M Ű E G Y E T E M 1 7 8 2

SZAKDOLGOZAT FELADAT

Marussy Kristóf
mérnök informatikus hallgató részére

Konfigurálható numerikus módszerek sztochasztikus modellekhez

A kritikus rendszerek – biztonságkritikus, elosztott és felhő-alapú alkalmazások – helyességének biztosításához szükséges a funkcionális és nemfunkcionális követelmények matematikai igényességű ellenőrzése. Számos, szolgáltatásbiztonsággal és teljesítményvizsgálattal kapcsolatos tipikus kérdés általában sztochasztikus analízis segítségével válaszolható meg.

A kritikus rendszerek elosztott és aszinkron tulajdonságai az állapottér robbanás jelenségéhez vezetnek. Emiatt méretük és komplexitásuk gyakran megakadályozza a sikeres sztochasztikus analízist, melynek számításigénye nagyban függ a lehetséges viselkedések számától. A modellek komponenseinek jellegzetes időbeli viselkedése és leginkább eltérő karakterisztikája a számításigény további jelentős növekedését okozhatja.

A szolgáltatásbiztonsági és teljesítményjellemzők kiszámítása markovi modellek állandósult állapotbeli és tranziens megoldását igényli. Számos eljárás ismert ezen problémák kezelésére, melyek eltérő reprezentációkat és numerikus algoritmusokat alkalmaznak; ám a modellek változatos tulajdonságai miatt nem választható ki olyan eljárás, mely minden esetben hatékony lenne. A hallgató feladata áttekinteni az irodalmat és megvizsgálni az ismert algoritmusokat.

A feladat megoldása a következő lépésekből áll:

1. Mutassa be az irodalomban ismert, markovi sztochasztikus rendszerek állandósult állapotbeli és tranziens viselkedésének vizsgálatára alkalmas numerikus algoritmusokat.
2. Az irodalom alapján implementáljon kiválasztott tranziens és állandósult állapotbeli analízis algoritmusokat.
3. Hasonlítsa össze futási idő és tárhely komplexitás szempontjából az implementált algoritmusokat.
4. Értékelje a megoldást és vizsgálja meg a továbbfejlesztési lehetőségeket.

Tanszéki konzulens:	Vörös András, tudományos segédmunkatárs
	Molnár Vince, doktorandusz
Külső konzulens:	dr. Telek Miklós, egyetemi tanár

Budapest, 2015. október 7.

Dr. Jobbágy Ákos
egyetemi tanár
tanszékvezető

Contents

Chapter 5

Algorithms for stochastic analysis

Steady state, transient, accumulated and sensitivity analysis problems pose several numerical challanges, especially when the state space of the CTMC and the vectors and matrices involved in the computation are extemely large.

In steady-state and sensitivty analysis, linear equations of the form $\mathbf{x}A = \mathbf{b}$ are solved, such as eqs. (2.2) and (2.6) on page 4 and on page 7. The steady-state probability vector is the solution of the linear system

$$\frac{\mathrm{d}\boldsymbol{\pi}}{\mathrm{d}t} = \boldsymbol{\pi} Q = \mathbf{0}, \quad \boldsymbol{\pi}\mathbf{1}^{\mathrm{T}} = 1, \tag{2.2 revisited}$$

where the infinitesimal generator Q is a rank-deficient matrix. Therefore, steady-state solution methods must handle various generator matrix decompositions and homogenous linear equation with rank deficient matrices. Convergence and computation times of linear equations solvers depend on the numerical properties of the Q matrices, thus different solvers may be preferred for different models.

In transient analysis, initial value problems with first-order linear differetial equations such as eqs. (2.1) and (2.5) on page 4 and on page 6 are considered. The decomposed generator matrix Q must be also handled efficiently. Another difficulty is caused by the *stiffness* of differential equations arising from some models, which may significantly increase computation times.

To facilitate configurable stochastic analysis, we developed several linear equation solvers and transient analysis methods. Where it is reasonable, the implementation is independent of the form of the generator matrix Q.

The implementation of low-level linear algebra operations is also decoupled from the numerical algorithms and data structure. This strategy enables further configurability by replacing the operations at runtime, as described in Chapter 4.

In this chapter, we describe the algorithms implemented in our stochastic analysis framework. The pseudocode of the algorithms is annotated with the low level operations performed on the configurable data structure by the high level algorithms.

Algorithm 5.1 Crout's LU decomposition without pivoting.

```
  Input: the matrix $A \in \mathbb{R}^{n \times n}$ operated on in-place
  Output: $L, U \in \mathbb{R}^{n \times n}$ such that $A = LU$, $u[i,i] = 1$ for all $i = 0, 1, \ldots, n-1$
1 for $i \leftarrow 0$ to $n-1$ do
2     for $j \leftarrow 0$ to $i$ do $a[i,j] \leftarrow a[i,j] - \sum_{k=0}^{j-1} a[i,k]a[k,j]$
3     for $j \leftarrow i+1$ to $n-1$ do $a[i,j] \leftarrow \big(a[i,j] - \sum_{k=0}^{i-1} a[i,k]a[i,j]\big) / a[i,i]$
4 Let $A_L$, $A_D$ and $A_U$ refer to the strictly lower triangular, diagonal and strictly
  upper triangular parts of $A$, respectively.
5 $L \leftarrow A_L + A_D$
6 $U \leftarrow A_U + I$
7 return $L, U$
```

5.1 Linear equation solvers

5.1.1 Explicit solution by LU decomposition

LU decomposition is a direct method for solving linear equations with forward and backward substitution, i.e. it does not require iteration to reach a given precision.

The decomposition computes the lower triangular matrix L and upper triangular matrix U such that

$$A = LU.$$

To solve the equation

$$\mathbf{x}A = \mathbf{x}LU = \mathbf{b}$$

forward substitution is applied first to find $\mathbf{z}$ in

$$\mathbf{z}U = \mathbf{b},$$

then $\mathbf{x}$ is computed by back substitution from

$$\mathbf{x}L = \mathbf{b}.$$

We used Crout's LU decomposition [11, Section 2.3.1], presented in Algorithm 5.1), which ensures

$$u[i,i] = 1 \text{ for all } i = 0, 1, \ldots, n-1,$$

i.e. the diagonal of the U matrix is uniformly 1. The matrix is filled in during the decomposition even if it was initially sparse, therefore it should first be copied to a dense array storage for efficiency reasons. This considerably limits the size of Markov chains that can be analysed by direct solution due to memory requirements. Our

Algorithm 5.2 Forward and back substitution.

```
Input: $U, L \in \mathbb{R}^{n \times n}$, right vector $\mathbf{b} \in \mathbb{R}^n$
Output: solution of $\mathbf{x}LU = \mathbf{b}$
1  allocate $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$
2  if $\mathbf{b} = \mathbf{0}$ then $\mathbf{z} \leftarrow \mathbf{0}$        // Skip forward substitution for homogenous equations
3  else for $j \leftarrow 0$ to $n-1$ do $z[j] \leftarrow b[j] \cdot \sum_{i=0}^{j-1} u[i,j]$
4  if $l[n-1,n-1] \approx 0$ then
5      if $z[n-1] \approx 0$ then $x[n-1] \leftarrow 0$        // Set the free parameter to 1
6      else error "inconsistent linear equation system"
7  else $x[n-1] \leftarrow z[n-1]/l[n-1,n-1]$
8  for $j \leftarrow n-2$ downto 0 do
9      if $l[j,j] \approx 0$ then error "more than one free parameter"
10     $x[j] \leftarrow \left(z[i] - \sum_{i=j+1}^{n-1} x[i]l[i,j]\right)/l[j,j]$
11 return $\mathbf{x}$
```

data structure allows access to upper and lower diagonal parts to matrices and linear combinations, therefore no additional storage is needed other than A itself.

The forward and back substitution process is shown in Algorithm 5.2. If multiple equations are solver with the same matrix, its LU decomposition may be cached.

Matrices of less than full rank

If the matrix Q is of rank $n-1$, the element $l[n-1,n-1]$ in Crout's LU decomposition will be 0. In this case, $x[n-1]$ is a free parameter and will be set to 1 to yield a nonzero solution vector when $z[n-1] = 0$. If $z[n-1] \neq 0$, the equation $\mathbf{x}L = \mathbf{z}$ does not have a solution and the error condition in line 6 is triggered. A matrix of rank less than $n-1$ triggers the error condition in line 9.

In practice, the algorithm can be used to solve homogenous equations in Markovian analysis, because the infinitesimal generator matrix Q of an irreducible CTMC is always of rank $n-1$. The solution vector $\mathbf{x}$ is not a probability vector in general, so it must be normalized as $\pi = \mathbf{x}/\mathbf{x}\mathbf{1}^{\mathrm{T}}$ to get a stationary probability distribution vector.

5.1.2 Iterative methods

Iterative methods express the solution of the linear equation $\mathbf{x}A = \mathbf{b}$ as a recurrence

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}),$$

Algorithm 5.3 Basic iterative scheme for solving linear equations.

```
Input: matrix $A \in \mathbb{R}^{n \times n}$, right vector $\mathbf{b} \in \mathbb{R}^n$, initial guess $\mathbf{x} \in \mathbb{R}^n$, tolerance $\tau > 0$
Output: approximate solution of $\mathbf{x}A = \mathbf{b}$ and its residual norm
1 allocate $\mathbf{x}' \in \mathbb{R}^n$                    // Previous iterate for convergence test
2 repeat
3     $\mathbf{x}' \leftarrow \mathbf{x}$                            // Save the previous vector
4     $\mathbf{x} \leftarrow f(\mathbf{x}')$
5 until $\|\mathbf{x}' - \mathbf{x}\| \leq \tau$
6 return $\mathbf{x}$ and $\|\mathbf{x}Q - \mathbf{b}\|$
```

where $\mathbf{x}_0$ is an initial guess vector. The iteration converges to a solution vector when $\lim_{k \to \infty} \mathbf{x}_k = \mathbf{x}$ exists and $\mathbf{x}$ equals the true solution vector $\mathbf{x}^*$. The iteration is illustrated in Algorithm 5.3.

The process is assumed to have converged if subsequent iterates are sufficiently close, i.e. the stopping criterion at the kth iteration is

$$\|\mathbf{x}_k - \mathbf{x}_{k-1}\| \leq \tau \tag{5.1}$$

for some prescribed tolerance τ. In our implementation, we selected the L^1-norm

$$\|\mathbf{x}_k - \mathbf{x}_{k-1}\| = \sum_i \left| x_k[i] - x_{k-1}[i] \right|$$

as the vector norm used for detecting convergence.

Premature termination may be avoided if iterates spaced $m > 1$ iterations apart are used for convergence test ($\|\mathbf{x}_k - \mathbf{x}_{k-m}\| \leq \tau$), but only at the expense of additional memory required for storing m previous iterates. In order to handle large Markov chains with reasonable memory consumption, we only used the convergence test with a single previous iterate.

Correctness of the solution can be checked by observing the norm of the residual $\mathbf{x}_k A - \mathbf{b}$, since the error vector $\mathbf{x}_k - \mathbf{x}^*$ is generally not available. Because the additional matrix multiplication may make the latter check costly, it is performed only after detecting convergence by eq. (5.1). Unfortunately, the residual norm may not be representative of the error norm if the problem is ill-conditioned.

For a detailed discussion stopping criterions and iterate normalization in steady-state CTMC analysis, we refer to [21, Section 10.3.5].

Algorithm 5.4 Power iteration.

```
1 $\alpha^{-1} \leftarrow 1/\max_i |a[i,i]|$
2 **repeat**
3     $\mathbf{x}' \leftarrow \mathbf{x}A$                          ▷ VectorMatrixMultiplyFromLeft
4     $\mathbf{x}' \leftarrow \mathbf{x}' + (-1) \cdot \mathbf{x}$            ▷ In-place VectorAdd
5     $\epsilon \leftarrow \alpha^{-1} \|\mathbf{x}'\|$                  ▷ VectorL1Norm
6     $\mathbf{x} \leftarrow \mathbf{x} + \alpha^{-1}\mathbf{x}'$                ▷ In-place VectorAdd
7 **until** $\epsilon \le \tau$
```

Power iteration

Power iteration [21, Section 10.3.1] is the one of the simplest iterative methods for Markovian analysis. Its iteration function has the form

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) = \mathbf{x}_{k-1} + \frac{1}{\alpha}(\mathbf{x}_{k-1}A - \mathbf{b}).$$

The iteration converges if the diagonal elements $a[i,i]$ of A are strictly negative, the off-diagonal elements $a[i,j]$ are nonnegative and $\alpha \ge \max_i |a[i,i]|$. The matrix A satisfies these properties if it is an inifinitesimal generator matrix of an irreducible CTMC. The fastest convergence is achieved when $\alpha = \min_i |a[i,i]|$.

Power iteration can be realized by replacing lines 2–5 in Algorithm 5.3 on page 32 with the loop in Algorithm 5.4.

This realization uses memory efficiently, because it only requires the allocation of a single vector $\mathbf{x}'$ in addition to the initial guess $\mathbf{x}$.

Observation 5.1 If $\mathbf{b} = 0$ and A is an inifitesimal generator matrix, then

$$\begin{aligned}\mathbf{x}_k \mathbf{1}^{\mathrm{T}} &= \left[\mathbf{x}_{k-1} + \frac{1}{\alpha}(\mathbf{x}_{k-1}A - \mathbf{b})\right]\mathbf{1}^{\mathrm{T}} \\ &= \mathbf{x}_{k-1}\mathbf{1}^{\mathrm{T}} + \frac{1}{\alpha}\mathbf{x}_{k-1}A\mathbf{1}^{T} - \mathbf{b}\mathbf{1}^{\mathrm{T}} \\ &= \mathbf{x}_{k-1}\mathbf{1}^{\mathrm{T}} + \frac{1}{\alpha}\mathbf{x}_{k-1}\mathbf{0}^{\mathrm{T}} - \mathbf{0}\mathbf{1}^{\mathrm{T}} = \mathbf{x}_{k-1}\mathbf{1}^{\mathrm{T}}.\end{aligned}$$

This means the sum of the elements of the result vector $\mathbf{x}$ and the initial guess vector $\mathbf{x}_0$ are equal, because the iteration leaves the sum unchanged.

To solve an equation of the form

$$\mathbf{x}Q = \mathbf{0}, \quad \mathbf{x}\mathbf{1}^{\mathrm{T}} = 1 \tag{5.2}$$

where Q is an infinitesimal generator matrix, the initial guess $\mathbf{x}_0$ is selected such that $\mathbf{x}_0 \mathbf{1}^\mathrm{T} = 1$. If the CTMC described by Q is irreducible, we may select

$$x_0[i] \equiv \frac{1}{n}, \tag{5.3}$$

where n is the dimensionality of $\mathbf{x}$. After the initial guess is selected, the equation $\mathbf{x}\mathbf{1}^\mathrm{T}$ may be ignored to solve $\mathbf{x}Q = \mathbf{0}$ with the power method. This process yields the solution of the original problem (5.2).

Jacobi and Gauss–Seidel iteration

Jordan and Gauss–Seidel iterative methods [21, Section 10.3.2–3] repeatedly solve a system of simultaneous equations of a specific form.

In Jordan iteration, the system

$$\left.\begin{aligned} b[0] &= x_k[0]a[0,0] + x_{k-1}[1]a[1,0] + \cdots + x_{k-1}[n-1]a[n-1,0], \\ b[1] &= x_{k-1}[0]a[0,1] + x_k[1]a[1,1] + \cdots + x_{k-1}[n-1]a[n-1,1], \\ &\vdots \\ b[n-1] &= x_{k-1}[0]a[0,n-1] + x_{k-1}[1]a[1,n-1] + \cdots + x_k[n-1]a[n-1,n-1], \end{aligned}\right\}$$

is solved for $\mathbf{x}_k$ at each iteration, i.e. there is a single unknown in each row and the rest of the variables are taken from the previous iterate. In vector form, the iteration can be expressed as

$$\mathbf{x}_k = A_D^{-1}(\mathbf{b} - A_O \mathbf{x}_{k-1}),$$

where A_D and A_O are the diagonal (all off-diagonal elements are zero) and off-diagonal (all diagonal elements are zero) parts of $A = A_D + A_O$.

In Gauss–Seidel iteration, the linear system

$$\left.\begin{aligned} b[0] &= x_k[0]a[0,0] + x_{k-1}[1]a[1,0] + \cdots + x_{k-1}[n-1]a[n-1,0], \\ b[1] &= x_k[0]a[0,1] + x_k[1]a[1,1] + \cdots + x_{k-1}[n-1]a[n-1,1], \\ &\vdots \\ b[n-1] &= x_k[0]a[0,n-1] + x_k[1]a[1,n-1] + \cdots + x_k[n-1]a[n-1,n-1], \end{aligned}\right\}$$

is considered, i.e. the ith equation contains the first i elements of $\mathbf{x}_k$ as unknowns. The equations are solved for successive elements of $\mathbf{x}_k$ from top to bottom.

Jacobi over-relaxation, a generalized form of Jacobi iteraion, is realized in Algorithm 5.5. The value 1 of the over-relaxation paramter ω corresponds to ordinary Jacobi iteration. Values $\omega > 1$ may accelerate convergence, while $0 < \omega < 1$ may help diverging Jacobi iteration converge.

Jacobi over-relaxation has many parallelization opportunities. The matrix multiplication in line 4 and the vector addition in line 5 can be parallelized, as well as the for loop in line 7. Our implementation takes advantage of the configurable linear algebra

Algorithm 5.5 Jacobi over-relaxation.

```
Input: matrix $A \in \mathbb{R}^{n \times n}$, right vector $\mathbf{b} \in \mathbb{R}^n$, initial guess $\mathbf{x} \in \mathbb{R}^n$, tolerance $\tau > 0$,
       over-relaxation parameter $\omega > 0$
Output: approximate solution of $\mathbf{x}A = \mathbf{b}$
allocate $\mathbf{x}' \in \mathbb{R}^n$
Let $A_O$ refer to the off-diagonal part of $A$.
repeat
    $\mathbf{x}' \leftarrow \mathbf{x}A_O$                              ▷ VectorMatrixMultiplyFromLeft
    $\mathbf{x}' \leftarrow \mathbf{x}' + (-1) \cdot \mathbf{b}$                     ▷ In-place VectorAdd
    $\epsilon \leftarrow 0$
    for $i \leftarrow 0$ to $n-1$ do
        $y \leftarrow (1-\omega)x[i] - \omega x'[i]/a[i,i]$
        $\epsilon \leftarrow \epsilon + |y - x[i]|$
        $x[i] \leftarrow y$
until $\epsilon \leq \tau$
return $\mathbf{x}$
```

operations framework to execute lines 4 and 5 with possible paralellization considering the structures of both the vectors $\mathbf{x}, \mathbf{x}'$ and the matrix A. However, the inner loop is left sequential to reduce implementation complexity, as it represents only a small fraction of execution time compared to the matrix-vector product.

Algorithm 5.6 shows an implementation of successive over-relaxation for Gauss–Seidel iteration, where the notation $\mathbf{a}_O[\cdot, i]$ refers to the ith column of A_O.

Gauss–Seidel iteration cannot easily be parallelized, because calculation of successive elements $x[0], x[1], \ldots$ depend on all of the prior elements. However, in contrast with Jacobi iteration, no memory is required in addition to the vectors $\mathbf{x}$, $\mathbf{b}$ and the matrix X, which makes the algorithm suitable for very large vectors and memory-constrained situations. In addition, convergence is often significantly faster.

The sum of elements $\mathbf{x1}^{\mathrm{T}}$ does not stay constant during Jacobi or Gauss–Seidel iteration. Thus, when solving equations of the form $\mathbf{x}Q = \mathbf{0}, \mathbf{x1}^{\mathrm{T}} = 1$, normalization cannot be entierly handled by the initial guess. We instead transform the equation into the form

$$\mathbf{x} \begin{pmatrix} q[0,0] & q[0,1] & \cdots & q[0,n-2] & 1 \\ q[1,0] & q[1,1] & \cdots & q[1,n-2] & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ q[n-2,0] & q[n-2,1] & \cdots & q[n-2,n-2] & 1 \\ q[n-1,0] & q[n-1,1] & \cdots & q[n-2,n-1] & 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}, \tag{5.4}$$

where we take advantage of the fact that the infinitesimal generator matrix is not of

Algorithm 5.6 Gauss–Seidel successive over-relaxatation.

```
Input: matrix $A \in \mathbb{R}^{n \times n}$, right vector $\mathbf{b} \in \mathbb{R}^n$, initial guess $\mathbf{x} \in \mathbb{R}^n$, tolerance $\tau > 0$,
       over-relaxation parameter $\omega > 0$
Output: approximate solution of $\mathbf{x}A = \mathbf{b}$
allocate $\mathbf{x}' \in \mathbb{R}^n$
Let $A_O$ refer to the off-diagonal part of $A$.
repeat
    $\epsilon \leftarrow 0$
    for $i \leftarrow 0$ to $n - 1$ do
        $scalarProduct \leftarrow \mathbf{x} \cdot \mathbf{a}_O[\cdot, i]$    ▷ VectorMatrixScalarProductWithColumn
        $y \leftarrow \omega(b[i] - scalarProduct)/a[i,i] + (1 - \omega) \cdot x[i]$
        $\epsilon \leftarrow \epsilon + |y - x[i]|$
        $x[i] \leftarrow y$
until $\epsilon \le \tau$
return $\mathbf{x}$
```

full rank, therefore one of the columns is redundant and can be replaced with the condition $\mathbf{x}\mathbf{1}^{\mathrm{T}} = 1$. While this transformation may affect the convergence behavior of the algorithm, it allows uniform handling of homogenous and non-homogenous linear equations.

5.1.3 Group iterative methods

Group or *block* iterative methods Stewart [21, Section 10.4] assume the block structure for the vectors $\mathbf{x}$, $\mathbf{b}$ and the matrix A

$$\mathbf{x}[i] \in \mathbb{R}^{n_i}, \mathbf{b}[j] \in \mathbb{R}^{n_j}, A[i,j] \in \mathbb{R}^{n_i \times n_j} \text{ for all } i, j \in \{0, 1, \dots, N-1\},$$

Infinitesimal generator matrices in the block Kronecker decomposition along with appropriately partitioned vectors match this structure (see eq. (2.10) on page 14). Each block of $\mathbf{x}$ corresponds to a group a variables that are simultaneously solved for.

Group Jacobi iteration solves the linear system

$$\left.\begin{aligned}
\mathbf{b}[0] &= \mathbf{x}_k[0]A[0,0] + \mathbf{x}_{k-1}[1]A[1,0] + \cdots + \mathbf{x}_{k-1}[n-1]A[n-1,0], \\
\mathbf{b}[1] &= \mathbf{x}_{k-1}[0]A[0,1] + \mathbf{x}_k[1]A[1,1] + \cdots + \mathbf{x}_{k-1}[n-1]A[n-1,1], \\
&\vdots \\
\mathbf{b}[n-1] &= \mathbf{x}_{k-1}[0]A[0,n-1] + \mathbf{x}_{k-1}[1]A[1,n-1] + \cdots + \mathbf{x}_k[n-1]A[n-1,n-1],
\end{aligned}\right\}$$

Algorithm 5.7 Group Jacobi over-relaxation.

```
   Input: block matrix $A$, block right vector $\mathbf{b}$, block initial guess $\mathbf{n}$, tolerance $\tau > 0$,
          over-relaxation parameter $\omega > 0$
   Output: approximate solution of $\mathbf{x}A = \mathbf{b}$ and its residual norm
 1 allocate $\mathbf{x}'$ and $\mathbf{c}$ with the same block structure as $\mathbf{x}$ and $\mathbf{b}$
 2 Let $A_{OB}$ represent the off-diagonal part of the block matrix $A$ with the blocks
   along the diagonal set to zero.
 3 repeat
 4     $\mathbf{x}' \leftarrow \mathbf{x}, \mathbf{c} \leftarrow \mathbf{b}$
 5     $\mathbf{c} \leftarrow \mathbf{c} + (-1) \cdot \mathbf{x}' A_{OB}$          ▷ AccumulateVectorMatrixMultiplyFromLeft
 6     parallel for $i \leftarrow 0$ to $N-1$ do                 // Loop over all blocks
 7         Solve $\mathbf{x}[i]A[i,i] = \mathbf{c}[i]$ for $\mathbf{x}[i]$
 8     $\epsilon \leftarrow 0$
 9     for $k \leftarrow 0$ to $n-1$ do                          // Loop over all elements
10         $y \leftarrow \omega x[k] + (1-\omega)x'[k]$
11         $\epsilon \leftarrow \epsilon + |y - x'[k]|$
12         $x[k] \leftarrow y$
13 until $\epsilon \le \tau$
```

while group Gauss–Seidel considers

$$\left.\begin{aligned}
\mathbf{b}[0] &= \mathbf{x}_k[0]A[0,0] + \mathbf{x}_{k-1}[1]A[1,0] + \cdots + \mathbf{x}_{k-1}[n-1]A[n-1,0], \\
\mathbf{b}[1] &= \mathbf{x}_k[0]A[0,1] + \mathbf{x}_k[1]A[1,1] + \cdots + \mathbf{x}_{k-1}[n-1]A[n-1,1], \\
&\vdots \\
\mathbf{b}[n-1] &= \mathbf{x}_k[0]A[0,n-1] + \mathbf{x}_k[1]A[1,n-1] + \cdots + \mathbf{x}_k[n-1]A[n-1,n-1].
\end{aligned}\right\}$$

Implementations of group Jacobi over-relaxation and group Gauss–Seidel successive over-relaxation are shown in Algorithms 5.7 and 5.8 on the current page and. The inner linear equations of the form $\mathbf{x}[i]A[i,i] = \mathbf{c}$ may be solved by any algorithm, for example, LU decomposition, iterative methods, or even block-iterative methods if A has a two-level block structure. The choice of the inner algorithm may significantly affect performance and care must be taken to avoid diverging inner solutions in an iterative solver is used.

In Jacobi over-relaxation, paralellization of both the matrix multiplication and the inner loop is possible. However, two vectors of the same size as $\mathbf{x}$ are required for temporary storage.

Gauss–Seidel successive over-relaxation cannot be parallelized easily. However it requires only two temporary vectors of size equal to the largest block of $\mathbf{x}$, much less than Jacobi over-relaxation. Moreover, it often requires fewer steps to converge, making

Algorithm 5.8 Group Gauss–Seidel successive over-relaxation.

```
Input: block matrix $A$, block right vector $\mathbf{b}$, block initial guess $\mathbf{n}$, tolerance $\tau > 0$,
       over-relaxation parameter $\omega > 0$
Output: approximate solution of $\mathbf{x}A = \mathbf{b}$ and its residual norm
allocate $\mathbf{x}'$ and $\mathbf{c}$ large enough to store a single block of $\mathbf{x}$ and $\mathbf{b}$.
repeat
    $\epsilon \leftarrow 0$
    for $i \leftarrow 0$ to $N-1$ do                                  // Loop over all blocks
        $\mathbf{x}' \leftarrow \mathbf{x}[i], \mathbf{c} \leftarrow \mathbf{b}[i]$
        for $j \leftarrow 0$ to $N-1$ do
            if $i \neq j$ then                  ▷ AccumulateVectorMatrixMultiplyFromLeft
                $\mathbf{c} \leftarrow \mathbf{c} + (-1) \cdot \mathbf{x}[j]A[i,j]$
        Solve $\mathbf{x}[i]A[i,i] = \mathbf{c}$ for $\mathbf{x}[i]$
        for $k \leftarrow 0$ to $n_i - 1$ do
            $y \leftarrow \omega x[i][k] + (1-\omega)x'[k]$
            $\epsilon \leftarrow \epsilon + |y - x'[k]|$
            $x[i][k] \leftarrow y$
until $\epsilon \leq \tau$
```

it preferable over Jacobi iteration.

Because the inner solver may be selected by the user and thus its convergence behaviour varies widely, we do not perform the transformation for homogenous equations (5.4). Instead, the normalization $\boldsymbol{\pi} = \mathbf{x}/\mathbf{x}\mathbf{1}^{\mathrm{T}}$ is performed only after finding any nonzero solution of $\mathbf{x}Q = \mathbf{0}$.

For a detailed analysis of the convergence behaviour of group iterative methods, we refer to Greenbaum [6, Chapter 14] and **TODO!!!**

5.1.4 Krylov subspace methods

Projectional iterative methods are iterative linear equation solvers that produce a sequence of approximate solutions $\mathbf{x}_k$ of the linear equation $\mathbf{x}A = \mathbf{b}$ that satisfy the Petrov–Galerkin conditions **TODO Cite**

$$\mathbf{x}_k \in \mathcal{K}_k, \quad \mathbf{r}_k = \mathbf{b} - \mathbf{x}_k A \perp \mathcal{L}_k, \tag{5.5}$$

where $\mathcal{K}_k$ and $\mathcal{L}_k$ are two subspaces of $\mathbb{R}^n$ and $\mathbf{r}_k$ is residual in the kth iteration.

Krylov subspace iterative methods correspond to the choice

$$\mathcal{K}_k = \mathcal{K}_k(A, \mathbf{r}_0) = \operatorname{span}\{\mathbf{r}_0, \mathbf{r}_0 A, \mathbf{r}_0 A^2, \ldots, \mathbf{r}_0 A^{k-1}\},$$

where $\mathcal{K}_k(A, \mathbf{r}_0)$ is the *kth Krylov subspace* of A and the initial residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{x}_0 Q$.

The smallest $m \in \mathbb{N}$ such that $\dim \mathcal{K}_m(A, \mathbf{r}_0) = \dim \mathcal{K}_{m+1}(A, \mathbf{r}_0)$ is called the *grade* of A with respect to $\mathbf{r}_0$. Hence $k \le m$ implies $\dim \mathcal{K}_k(A, \mathbf{r}_0) = k$. Krylov subspace solvers usually suppose that the algorithm terminates at some iteration k^* such that $k^* \le m$, therefore the dimension of $\mathcal{K}_k$ increases with each iteration. The contary situation leads to stagnation, because $\mathcal{K}_k \subseteq \mathcal{K}_{k+1}$ together with $\dim \mathcal{K}_k = \dim \mathcal{K}_{k+1}$ $(k \ge m)$ implies $\mathcal{K}_k = \mathcal{K}_{k+1}$.

The subspace $\mathcal{L}_k$ also must be a k-dimensional subspace of $\mathbb{R}^n$. Conceptually, while the Krylov subspace $\mathcal{K}_k$ "expands" in dimensionality every iteration, the subspace $\mathcal{L}_k$ likewise fills the space to make additional residuals forbidden by the Petrov–Galerkin condition (5.5).

If $A \in \mathbb{R}^{n \times n}$ is of full rank and grade, Krylov subspace solvers find the exact solution of the linear equation in at most n iterations with exact arithmetic. The only possible orthogonal residual is the zero vector $\mathbf{0}$ if $\mathcal{L}_n = \mathbb{R}^n$ holds. While n is usually too large for this to be practical, convergence often happens with suitable accuracy after a small number of iterations.

Note that problems may arise when A is singular, which may worsen the convergence behaviour. This is the case in CTMC analysis, where the infinitesimal generator matrix Q is of rank $n - 1$.

Some Krylov subspace methods for nonsymmetric matrices in wide use are Generalized Minimum Residual (GMRES) [14], Bi-Conjugate Gradient Stabilized (BiCGSTAB) [23], Conjugate Gradient Squared (CGS) [19] and IDR(s) [20].

Generalized Minimal Residual (GMRES)

Generalized Minimal Residual (GMRES) [15, Section 6.5.1; 14] a Krylov subspace method for nonsymmetric linear systems. It is based on the choice

$$\mathcal{L}_k = \mathcal{K}_k A = \{\mathbf{r}_0 A, \mathbf{r}_0 A^2, \ldots, \mathbf{r}_0 A^k\}.$$

With this choice, the Petrov–Galerkin condition (5.5) minimizes the Euclidean norm of the residuals in each iteration, i.e.

$$\mathbf{x}_k \in \mathcal{K}_k \text{ such that } \mathbf{r}_k = \mathbf{b} - \mathbf{x}_k A \perp \mathcal{L}_k \quad \Longleftrightarrow \quad \mathbf{x}_k = \operatorname*{arg\,min}_{\mathbf{x} \in \mathcal{K}_k} \|\mathbf{b} - \mathbf{x} Q\|_2. \tag{5.6}$$

Unfortunately, the solution of eq. (5.6) requires the storage of a basis of $\mathcal{K}_k$, which is a k dimensional subspace of $\mathbb{R}^n$. Thus, each iteration requires the allocation of an additional vector. Solution of a linear system with GMRES requires up to n additional floating-point vectors of n elements each, i.e. $O(n^2)$ floating-point numbers. This property makes GMRES a "long recurrence" algorithm.

The high memory requirements may be alleviated by discarding the basis of $\mathcal{K}_k$ and restarting the iteration from another initial guess $\mathbf{x}_0$ if no solution is obtained after ℓ iterations. The resulting algorithm is called GMRES(ℓ).

The convergence behaviour of full GMRES is often excellent. However, due to impractical memory requirements, we did not implement GMRES as a numerical solver in our framefork. We instead use BiCGSTAB and IDR(s)STAB(ℓ), Krylov subspace solvers incorporating GMRES(ℓ)-like steps.

Bi-Conjugate Gradient Stabilized (BiCGSTAB)

Bi-Conjugate Gradient Stabilized (BiCGSTAB) [15, Section 7.4.2; 23] is a Krylov subspace method where [16]

$$\mathcal{L}_k = \mathcal{K}_k(A^{\mathrm{T}}, \tilde{\mathbf{r}}_0) \cdot (\Omega_k(A)^{\mathrm{T}})^{-1}, \quad \Omega_k(A) = \begin{cases} \Omega_{k-1}(A) \cdot (I - \omega_k A) & \text{if } k \geq 1, \\ I & \text{if } k = 0. \end{cases} \tag{5.7}$$

The *initial shadow residual* $\tilde{\mathbf{r}}_0$ must satisfy $\mathbf{r}_0 \tilde{\mathbf{r}}_0^{\mathrm{T}} \neq 0$ and must not be an eigenvector of Q^{T}. Usually, $\tilde{\mathbf{r}}_0 = \mathbf{r}_0$, which is the convention we use in our implementation.

Equivalently, BiCGSTAB is a Krylov subspace method which produces residuals

$$\mathbf{r}_k \in \mathcal{G}_k, \quad \mathcal{G}_k = \begin{cases} (\mathcal{G}_k \cap \tilde{\mathbf{r}}_0^{\perp})(I - \omega_k A) & \text{if } k \geq 1, \\ \mathbb{R}^n & \text{if } k = 0, \end{cases} \tag{5.8}$$

where $\mathcal{A}^{\perp}$ is the set of vector orthogonal to $\mathcal{A}$. It can be shown that [18]

$$\mathcal{G}_k = \mathcal{S}(\Omega_k, A, \tilde{\mathbf{r}}_0) = \{\mathbf{v} \cdot \Omega_k(A) : \mathbf{v} \perp \mathcal{K}_k(A^{\mathrm{T}}, \tilde{\mathbf{r}}_0)\},$$

where $\mathcal{S}(\Omega, A, \tilde{\mathbf{r}}_0)$ is the Ωth *Sonneveld subspace* generated by A and $\tilde{\mathbf{r}}_0$) of order $k = \deg \Omega$. Hence $\mathcal{L}_k = \mathcal{G}_k^{\perp}$, which makes BiCGSTAB equivalent to another Krylov subspace method, Induced Dimensionality Reduction (IDR) [18] in exact arithmetic.

BiCGSTAB is a "short recurrence", that is, the number of allocated intermediate vectors does not depend on the number of variables in equation system.

Implementation We selected BiCGSTAB as the first Krylov subspace solver integrated into our framework because of its good convergence behaviour and low memory requirements. BiCGSTAB only requires the storage of 7 vectors, which makes it suitable even for large state spaces with large state vectors.

Algorithm 5.9 shows the pseudocode for BiCGSTAB. Our implementation is based on the MATLAB code[1] by Barrett et al. [3].

[1] `http://www.netlib.org/templates/matlab/bicgstab.m`

Algorithm 5.9 BiCGSTAB iteration without preconditioning.

```
Input: matrix $A \in \mathbb{R}^{n\times n}$, right vector $\mathbf{b} \in \mathbb{R}^n$, initial guess $\mathbf{x} \in \mathbb{R}^n$, tolerance $\tau > 0$
Output: approximate solution of $\mathbf{x}A = \mathbf{b}$
allocate $\mathbf{r}, \tilde{\mathbf{r}}_0, \mathbf{v}, \mathbf{p}, \mathbf{s}, \mathbf{t} \in \mathbb{R}^n$
$\mathbf{r} \leftarrow \mathbf{b}$                                          ▷ VectorSet
$\mathbf{r} \leftarrow \mathbf{r} + (-1)\cdot \mathbf{x}A$                  ▷ VectorMatrixAccumulateMultiplyFromLeft
if $\|\mathbf{r}\| \le \tau$ then
    message "initial guess is correct, skipping iteration"
    return x
$\tilde{\mathbf{r}}_0 \leftarrow \mathbf{r}, \mathbf{v} \leftarrow \mathbf{0}, \mathbf{p} \leftarrow \mathbf{0}, \rho' \leftarrow 1, \alpha \leftarrow 1, \omega \leftarrow 1$
while true do
    Bi-CG step
    $\rho \leftarrow \mathbf{r}_0 \cdot \mathbf{r}$                        ▷ VectorScalarProduct
    if $\rho \approx 0$ then error "breakdown: $\mathbf{r} \perp \tilde{\mathbf{r}}_0$"
    $\beta \leftarrow \rho/\rho' \cdot \alpha/\omega$
    $\mathbf{p} \leftarrow \mathbf{r} + \beta \cdot \mathbf{p}$              ▷ VectorAdd
    $\mathbf{p} \leftarrow \mathbf{p} + (-\beta\omega)\cdot \mathbf{v}$      ▷ In-place VectorAdd
    $\mathbf{v} \leftarrow \mathbf{p}Q$                                      ▷ VectorMatrixMultiplyFromLeft
    $\alpha \leftarrow \rho/(\tilde{\mathbf{r}}_0 \cdot \mathbf{v})$          ▷ ScalarProduct
    $\mathbf{r} \leftarrow \mathbf{s} + (-\alpha)\cdot \mathbf{s}$           ▷ VectorAdd
    if $\|\mathbf{s}\| < \tau$ then
        $\mathbf{x} \leftarrow \mathbf{x} + \alpha \cdot \mathbf{p}$         ▷ In-place VectorAdd
        message "early return with vanishing s"
        return x
    GMRES(1) step
    $\mathbf{t} \leftarrow \mathbf{s}A$                                      ▷ VectorMatrixMultiplyFromLeft
    $tLengthSquared \leftarrow \mathbf{t}\cdot\mathbf{t}$                    ▷ ScalarProduct
    if $tLengthSquared \approx 0$ then error "breakdown: $\mathbf{t} \approx \mathbf{0}$"
    $\omega \leftarrow (\mathbf{t}\cdot\mathbf{s})/tLengthSquared$            ▷ ScalarProduct
    if $\omega \approx 0$ then error "breakdown: $\omega \approx 0$"
    $\epsilon \leftarrow 0$
    for $i \leftarrow 0$ to $n-1$ do
        $change \leftarrow \alpha p[i] + \omega s[i]$, $\epsilon \leftarrow \epsilon + |change|$. $x[i] \leftarrow x[i] + change$
    if $\epsilon \le \tau$ then return x
    $\mathbf{s} \leftarrow \mathbf{t} + (-\omega)\cdot \mathbf{r}$           ▷ VectorAdd
    $\rho' \leftarrow \rho$
```

The inner loop of BiCGSTAB is the composed of two procedures. The bi-conjucate gradient (Bi-CG) part in lines 9–20 calculates a residual $\mathbf{t} \in \mathcal{G}_{k-1}A$ and its associated approximate solution $\mathbf{x} \in \mathcal{K}_k$. The GMRES(1) part in lines 21–31 selects $\omega_k \in \mathbb{R}$ and calculates a new residual $\mathbf{r} \in \mathcal{G}_k$ such that the Euclidean norm $\|\mathbf{r}\|_2$ is minimized. This part improves convergence over the original Bi-Conjucate Gradient algorithm.

Solving preconditioned equations in the form $\mathbf{x}AM^{-1} = \mathbf{b}M^{-1}$ could improve convergence, but was omitted from our current implementation. As the choice of appropriate preconditioner matrices M is not trivial [8], implementation and sudy of preconditioners for Markov chains, especially with block Kronecker decomposition, is in the scope of our future work.

Because six vectors are allocated in addition to $\mathbf{x}$ and $\mathbf{b}$, the amount of available memory may be a significant bottleneck.

Similar to Observation 5.1 on page 33, it can be seen that the sum $\mathbf{x}\mathbf{1}^{\mathrm{T}}$ stays constant throughout BiCGSTAB iteration. Thus, we can find probability vectors satisfying homogenous equations by the initialization in eq. (5.3) on page 34.

Induced Dimensionality Reduction Stabilized (IDRSTAB)

Induced Dimensionality Reduction Stabilized (IDR(s)STAB(ℓ)) [18] is Krylov subspace solver that generalizes BiCGSTAB and IDR techniques to provide converge behaviors closely matching GMRES while maintaining the short recurrence property.

As the algorithm developed relatively recently in 2010, high performance implementations of IDR(s)STAB(ℓ) are not widely available. To our best knowledge, IDR(s)STAB(ℓ) was not investigated for use in CTMC analysis despite its promising results solving differential equations arising from finite element problems. Therefore, we are currently focusing research and development effort into integrating IDR(s)STAB(ℓ) into our stochastic analysis. Special attention is paid to its behaviour on steady-state equations with infinitesimal generator matrices and other linear systems arising from CTMC analysis.

IDR(s)STAB(ℓ) merges two generalizations of BiCGSTAB:

- The first idea comes from IDR(s) [20], a Krylov subspace solver based on Sonneveld subspaces. A block version of eq. (5.8) constraints the residual $\mathbf{r}_k$

  $$\mathbf{r}_k \in \mathcal{G}_k = \mathcal{S}(\Omega_k, A, \widetilde{R}_0) = \{\mathbf{v} \cdot \Omega_k(A) : \mathbf{v} \perp \mathcal{K}_k(A, \tilde{R}_0)\},$$

  where $\mathcal{K}_k(A, \tilde{R}_0)$ is the kth *row* Krylov subspace of $A \in \mathbb{R}^{n \times n}$ with respect to $\tilde{R}_0 \in \mathbb{R}^{s \times n}$

  $$\mathcal{K}_k(A, \widetilde{R}_0) = \operatorname{span}\{\tilde{\mathbf{r}}_0[i], \tilde{\mathbf{r}}_0[i]A, \ldots, \tilde{\mathbf{r}}_0[i]A^{k-1} : i = 0, 1, \ldots, s-1\}$$

  and $\tilde{\mathbf{r}}_0[i]$ is the ith row $\tilde{R}_0$.

Higher values of s, i.e. higher dimensional initial shadow spaces, may accelerate convergence, at the cost of allocation additional intermediate vectors.

- The second generalization, which is called BiCGSTAB(ℓ) [17], replaces the stabilizer polynomial Ω_k from eq. (5.7) with

  $$\Omega_k(A) = \Omega_{k-1}(A) \cdot (I - \gamma[0]A - \gamma[1]A^2 - \cdots - \gamma[\ell-1]A^\ell),$$

  i.e. degree of the stabilizer polynomial Ω increases by ℓ instead of 1 every iteration. The increase is described by the vector $\vec{\gamma} \in \mathbb{R}^\ell$.

  The higher-order stabilization, also called a GMRES(ℓ) step, improves convergence behavior with unsymmetrix matrices that have complex spectrum. However, the number of intermediate vectors, thus the amount of required memory, also grows.

A single dimensional initial shadow space ($s = 1$) and first-order stabilization ($\ell = 1$) make IDR(s)STAB(ℓ) identical to BiCGSTAB. Moreover, $\ell = 1$ results is behavior equivalent to IDR(s), while $s = 1$ results in behavior equivalent to BiCGSTAB(ℓ).

These correspondences make IDR(s)STAB(ℓ) a promising candidate for use in configurable stochastic analysis, as different settings of (s, ℓ) bring the power of multiple algorithms to the modelers' disposal.

Implementation The pseudocode of our implementation of IDR(s)STAB(ℓ), which is based on the pseudocode of Sleijpen and Van Gijzen [18], is show in **TODO**.

For convenient representation of memory requirements, we employ two different typographical styles for vectors. Vectors in bold, e.g. $\mathbf{x} \in \mathbb{R}^n$ are "long" vectors, while vectors with arrows, e.g. $\vec{\gamma}$ are "short" vectors of length s or $\ell \ll n$. Storage space of long vectors dominated memory requirements and their manipulations including vector–matrix products dominate computation time.

The algorithm works with three arrays of vectors, $\mathbf{R} \in \mathbb{R}^{(\ell+1)\times n}$, $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{((\ell+2)\times s)\times n}$, i.e. $\mathbf{r}[j'], \mathbf{u}[j,q], \mathbf{v}[j,q] \in \mathbb{R}^n$ for all $j' = 0, 1, \ldots, \ell$; $j = 0, 1, \ldots, \ell + 1$; $q = 0, 1, \ldots, s - 1$.

Algorithm 5.10 IDR(s)STAB(ℓ).

```
   Input: matrix $A \in \mathbb{R}^{n \times n}$, right vector $\mathbf{b} \in \mathbb{R}^n$, initial guess $\mathbf{x} \in \mathbb{R}^n$, tolerance $\tau > 0$
   Output: approximate solution of $\mathbf{x}A = \mathbf{b}$
1  allocate $\mathbf{R} \in \mathbb{R}^{(\ell+1) \times n}, \mathbf{U}, \mathbf{V} \in \mathbb{R}^{((\ell+2) \times s) \times n}, \widetilde{\mathbf{R}}_0 \in \mathbb{R}^{s \times n}$
2  allocate $\sigma \in \mathbb{R}^{s \times s}, \vec{m} \in \mathbb{R}^s, \vec{\alpha} \in \mathbb{R}^s, \vec{\beta} \in \mathbb{R}^s, G \in \mathbb{R}^{\ell \times \ell}, \vec{\rho} \in \mathbb{R}^\ell, \vec{\gamma} \in \mathbb{R}^\ell$
3  allocate $\mathbf{C}, \mathbf{D} \in \mathbb{R}^{s \times n}$

   Initialize shadow residuals

4  for $j = 0$ to $s - 1$ do
5      Sample $\tilde{\mathbf{r}}_0[j]$ from an $n$-dimensional standard normal distribution
6*     $\tilde{\mathbf{r}}_0[j] \leftarrow \tilde{\mathbf{r}}_0[j] + \left(-\sum_{k=0}^{n-1} \tilde{r}_0[j][k]/n\right) \mathbf{1}$     ▷ In-place VectorAdd
7      for $q = 0$ to $s - 1$ do
8          $\tilde{\mathbf{r}}_0[j] \leftarrow \tilde{\mathbf{r}}_0[j] + (-\tilde{\mathbf{r}}_0[j] \cdot \tilde{\mathbf{r}}_0[q]) \tilde{\mathbf{r}}_0[q]$     ▷ In-place VectorAdd
9      $\tilde{\mathbf{r}}_0[j] \leftarrow (1/\|\tilde{\mathbf{r}}_0[j]\|_2) \tilde{\mathbf{r}}_0[j]$     ▷ In-place VectorScale

   Initialize residuals and intermediate vectors

10 $\mathbf{r}[0] \leftarrow \mathbf{b}$, $\mathbf{r}[0] \leftarrow \mathbf{r}[0] + (-1)\mathbf{x}A$     ▷ MatrixVectorAccumulateMultiplyFromLeft
11 for $q \leftarrow 0$ to $s - 1$ do
12     if $q = 0$ then $\mathbf{c}[0] \leftarrow (-1)\mathbf{x}$, $\mathbf{u}[0,0] \leftarrow \mathbf{r}[0]$     ▷ VectorSet
13     else $\mathbf{c}[q] \leftarrow \mathbf{u}[0, q-1]$, $\mathbf{u}[0,q] \leftarrow \mathbf{u}[1, q-1]$     ▷ VectorSet
14     $U[1,q] \leftarrow U[0,q]A$     ▷ MatrixMultiplyFromLeft
15     for $k \leftarrow 0$ to $q - 1$ do
16         $proj \leftarrow \mathbf{u}[0,k] \cdot \mathbf{u}[0,q]$     ▷ VectorScalarProduct
17         $\mathbf{c}[q] \leftarrow \mathbf{c}[q] + (-proj)\,\mathbf{c}[k]$     ▷ In-place VectorAdd
18         $\mathbf{u}[0,q] \leftarrow U[0,q] + (-proj)\,\mathbf{u}[0,k]$, $\mathbf{u}[1,q] \leftarrow U[1,q] + (-proj)\,\mathbf{u}[1,k]$
19     $norm \leftarrow \|\mathbf{u}[0,q]\|_2$     ▷ VectorScalarProduct
20     $\mathbf{c}[q] \leftarrow (1/norm)\,\mathbf{c}[q]$,     ▷ In-place VectorScale
21     $\mathbf{u}[0,q] \leftarrow (1/norm)\,\mathbf{u}[0,q]$, $\mathbf{u}[1,q] \leftarrow (1/norm)\,\mathbf{u}[1,q]$

   Iteration

22 while $\|\mathbf{r}[0]\| > \epsilon$ do
23     Perform IDR step from TODO
24     Perform GMRES($\ell$) step from TODO
25 return $\mathbf{x}$
```

Algorithm 5.11 IDR(s)STAB(ℓ) IDR step.

```
for $j \leftarrow 1$ to $\ell$ do                                          // For every repetition step
    for $k \leftarrow 0$ to $s-1$ do
        for $q \leftarrow 0$ to $s-1$ do $\sigma[q,k] \leftarrow \mathbf{u}[j,q] \cdot \tilde{\mathbf{r}}_0[k]$     ▷ VectorScalarProduct
        $m[k] \leftarrow \mathbf{r}[j-1] \cdot \tilde{\mathbf{r}}_0[k]$                    ▷ VectorScalarProduct
    Solve $\vec{\alpha}\sigma = \vec{m}$ for $\vec{\alpha}$
    for $q \leftarrow 0$ to $s-1$ do
        $\mathbf{x} \leftarrow \mathbf{x} + \alpha[q]\,\mathbf{u}[0,q]$                             ▷ In-place VectorAdd
        for $k \leftarrow 0$ to $j-1$ do $\mathbf{r}[k] \leftarrow \mathbf{r}[k] + (-\alpha[q])\,\mathbf{u}[k+1][q]$
    $\mathbf{r}[j-1] \leftarrow \mathbf{r}A$                                  ▷ VectorMatrixMultiplyFromLeft
    for $q \leftarrow 0$ to $s-1$ do                      // Build a new basis for the shadow space
        if $q = 0$ then
            $\mathbf{d}[0] \leftarrow (-1)\,\mathbf{x}$                                          ▷ VectorScale
            for $k \leftarrow 0$ to $j-1$ do $\mathbf{v}[k,0] \leftarrow \mathbf{r}[k]$                   ▷ VectorSet
        else
            $\mathbf{d}[q] \leftarrow \mathbf{v}[0,q-1]$                                       ▷ VectorSet
            for $k \leftarrow 0$ to $j-1$ do $\mathbf{v}[k,q] \leftarrow \mathbf{v}[k+1,q-1]$             ▷ VectorSet
        for $k \leftarrow 0$ to $s-1$ do $m[k] \leftarrow \mathbf{v}[j,q] \cdot \tilde{\mathbf{r}}_0[k]$       ▷ VectorScalarProduct
        Solve $\vec{\beta}\sigma = \vec{m}$ for $\vec{\beta}$
        for $i \leftarrow 0$ to $s-1$ do
            $\mathbf{d}[q] \leftarrow \mathbf{d}[q] + (-\beta[i])\,\mathbf{c}[i]$                        ▷ In-place VectorAdd
            for $k \leftarrow 0$ to $j$ do $\mathbf{v}[k,q] \leftarrow \mathbf{v}[k,q] + (-\beta[i])\,\mathbf{u}[k,i]$
        $\mathbf{v}[j+1,q] \leftarrow \mathbf{v}[j,q]A$                           ▷ VectorMatrixMultiplyFromLeft
        for $i \leftarrow 0$ to $q-1$ do                          // Attempt orthonormalization
            $proj \leftarrow \mathbf{v}[j,q] \cdot \mathbf{v}[j,i]$                              ▷ VectorScalarProduct
            $\mathbf{d}[q] \leftarrow \mathbf{d}[q] + (-proj)\,\mathbf{d}[i]$                          ▷ In-place VectorAdd
            for $k \leftarrow 0$ to $j+1$ do $\mathbf{v}[k,q] \leftarrow \mathbf{v}[k,q] + (-proj)\,\mathbf{v}[k,i]$
        $norm \leftarrow \|\mathbf{v}[j,q]\|_2$                                  ▷ VectorScalarProduct
        if $norm < \epsilon$ then                                 // Gram–Schmidt breakdown
            message "early exit with $\mathbf{v}[j,q] \approx \mathbf{0}$"
            $sum \leftarrow \sum_{k=0}^{n-1} d[q][k]$, $\mathbf{x} \leftarrow (1/sum)\,d[q][k]$     ▷ In-place VectorScale
            return $\mathbf{x}$
        $\mathbf{d}[q] \leftarrow (1/norm)\,\mathbf{d}[q]$                                ▷ In-place VectorScale
        for $k \leftarrow 0$ to $j+1$ do $\mathbf{v}[k,q] \leftarrow (1/norm)\,\mathbf{v}[k,q]$      ▷ In-place VectorScale
    Swap the references to $\mathbf{C}$ and $\mathbf{D}$
    Swap the references to $\mathbf{U}$ and $\mathbf{V}$
```

Algorithm 5.12 IDR(s)STAB(ℓ) GMRES(ℓ) step.

```
Find $\arg\min_{\vec{\gamma}\in\mathbb{R}^\ell} \|\mathbf{r}[0] - R[1:\ell]\vec{\gamma}^{\mathrm{T}}\|_2$ by solving the normal equation
1  for $j \leftarrow 1$ to $s$ do
2      for $i \leftarrow 1$ to $s$ do $g[i,j] \leftarrow \mathbf{r}[i]\cdot\mathbf{r}[j]$          ▷ VectorScalarProduct
3      $\rho[j] \leftarrow \mathbf{r}[0]\cdot\mathbf{r}[j]$                                  ▷ VectorScalarProduct
4  Solve $\vec{\gamma}G = \vec{\rho}$ for $\vec{\gamma}$

Calculate the minimal residual
5  for $j \leftarrow 0$ to $j-1$ do
6      $\mathbf{x} \leftarrow \mathbf{x} + \gamma[j]\mathbf{r}[j]$                              ▷ In-place VectorAdd
7      $\mathbf{r}[0] \leftarrow \mathbf{r}[0] + (-\gamma[j])\mathbf{r}[j+1]$                     ▷ In-place VectorAdd
8      for $q \leftarrow 0$ to $s-1$ do
9          $\mathbf{c}[q] \leftarrow \mathbf{c}[q] + (-\gamma[j])\mathbf{u}[j,q]$                  ▷ In-place VectorAdd
10         $\mathbf{u}[j,q] \leftarrow \mathbf{u}[j,q] + (-\gamma[j])\mathbf{u}[j+1,q]$            ▷ In-place VectorAdd
11         $\mathbf{u}[j+1,q] \leftarrow \mathbf{u}[j+1,q] + (-\gamma[j])\mathbf{u}[j+2,q]$        ▷ In-place VectorAdd
```

The IDR part performs the projections, called a "repetition step",

$$\begin{tikzcd}
\mathbf{x}_- \arrow[r] & \mathbf{x} \arrow[d] & \mathbf{v}[0,0] \arrow[d, "A"] & \mathbf{v}[0,1] \arrow[d, "A"] & \cdots & \mathbf{v}[0,s-1], \arrow[d, "A"] \\
\mathbf{r}_-[0] \arrow[r, "\Pi_1"] & \mathbf{r}[0] \arrow[d, "A"] \arrow[ur, "\Pi_0"] & \mathbf{v}[1,0] \arrow[d, "A"] \arrow[ur, "\Pi_0"] & \mathbf{v}[1,1] \arrow[d, "A"] & \cdots & \mathbf{v}[1,s-1], \arrow[d, "A"] \\
\mathbf{r}_-[1] \arrow[r, "\Pi_2"] & \mathbf{r}[1] \arrow[ur, "\Pi_1"] & \mathbf{v}[2,0] \arrow[ur, "\Pi_1"] & \mathbf{v}[2,1] & \cdots & \mathbf{v}[2,s-1], \\
\vdots & \vdots & \vdots & \vdots & & \vdots \\
\mathbf{r}_-[j-2] \arrow[r, "\Pi_{j-1}"] & \mathbf{r}[j-2] \arrow[d, "A"] & \mathbf{v}[j-1,0] \arrow[d, "A"] & \mathbf{v}[j-1,1] \arrow[d, "A"] & \cdots & \mathbf{v}[j-1,s-1], \arrow[d, "A"] \\
\mathbf{r}_-[j-1] \arrow[r, "\Pi_j"] & \mathbf{r}[j-1] \arrow[d, "A"] \arrow[ur, "\Pi_{j-1}"] & \mathbf{v}[j,0] \arrow[d, "A"] \arrow[ur, "\Pi_{j-1}"] & \mathbf{v}[j,1] \arrow[d, "A"] & \cdots & \mathbf{v}[j,s-1], \arrow[d, "A"] \\
 & \boxed{\mathbf{r}[j]} \arrow[ur, "\Pi_j"] & \boxed{\mathbf{v}[j+1,0]} \arrow[ur, "\Pi_j"] & \boxed{\mathbf{v}[j+1,1]} & \cdots & \boxed{\mathbf{v}[j+1,s-1]}
\end{tikzcd} \tag{5.9}$$

for $j = 1, 2, \ldots, \ell$ every iteration.

After a repetition step is complete, **U** and **V** are swapped and the process starts again with increased j or the GMRES(ℓ) part commences. Symbols with a subscript "–" sign refer to vectors from the previous repetition step.

The projections Π_i $(i = 0, 1, \ldots, j)$ are defined as

$$\Pi_i = I - A^{j-i}\widetilde{R}_0\sigma^{-1}(U[i,\cdot])^{\mathrm{T}}, \quad \sigma = \widetilde{R}_0(U[j,\cdot])^{\mathrm{T}},$$

where the matrix $U[k,\cdot]$ is the $s \times n$ matrix that has the vector $\mathbf{u}[k,q]$ as its qth row. They ensure that the rows of $U[j,\cdot]$ form a basis of the Krylov subspace $\mathcal{K}_s(A\Pi_j, \mathbf{r}[j]\Pi_j)$ after the jth repetition step.

The relationships $\mathbf{r}[i+1] = \mathbf{r}iA$, $\mathbf{u}[i+1,q] = \mathbf{u}[i,q]A$, $\mathbf{v}[i+1,q] = \mathbf{v}[i,q]$ are maintained throughout the algorithm via the projections. This is signified by the gray $\downarrow A$ arrows in eq. (5.9). Notice that this means $\mathbf{r}[0]A^i = \mathbf{r}[i]$ and $U[0,\cdot]A^i = U[i,\cdot]$.

In the case of $\mathbf{r}[j]$ and $\mathbf{v}[j+1,q]$, a matrix multiplication is performed as shown by the $\downarrow A$ arrows. Vectors generated by matrix multiplication are shown in borders.

For improving numerical properties, Sleijpen and Van Gijzen [18] recommend performing Gram–Schmidt ortonormalization on $U[j,\cdot]$. The same subtractions and normalization operations must be performed on the rows of $U[i,\cdot]$, $i \neq j$ that are performed in $U[j,\cdot]$ in order to maintain their relationships. We realized the orthonormalization by the modified Gram–Schmidt process in lines **TODO** and **TODO**.

The storage of **R**, **U** and **V** requires $(\ell+1) + 2\cdot(\ell+1)\cdot s$ vectors of length n, while the initial shadow residual matrix $\widetilde{R}_0$ requires space equal to s vectors of length n. Thus, $1+\ell+3s+2\ell s$ intermediate vectors are needed in addition to the initial guess $\mathbf{x}_0$ and the right vectors $\mathbf{b}$. Although the memory requiremens of IDR(s)STAB(ℓ) are quite high, s and ℓ can be selected to ensure that the solution fits in the available memory.

A different formulation of the IDR(s)STAB(ℓ) principles is GBi-CGSTAB(s,ℓ) [22], which avoids the allocation of **V** by updating **U** in place, albeit with lesser numerical properties due to the lack of orthonormalization steps. Another variant by Aihara et al. [1] replaces some vector updates with matrix multiplications to improve accuracy.

Numerical problems and breakdown **TODO Leiras, diagram**

5.2 Transient analysis

5.2.1 Uniformization

The *uniformization* or *randomization* method solves the initial value problem

$$\frac{\mathrm{d}\pi(t)}{\mathrm{d}t} = \pi(t)Q, \quad \pi(t) = \pi 0 \tag{2.1 revisited}$$

by computing

$$\pi(t) = \sum_{k=0}^{\infty} \pi_0 P^k e^{-\alpha t}\frac{(\alpha t)^k}{k!}, \tag{5.10}$$

where $P = \alpha^{-1}Q+I$, $\alpha \geq \max_i |a[i,i]|$ and $e^{-\alpha t}\frac{(\alpha t)^k}{k!}$ is the value of the Poisson probabilty function with rate αt at k.

Integrating both sides of eq. (5.10) on page 47 to compute $\mathbf{L}(t)$ yields [13]

$$\begin{aligned}\int_0^t \boldsymbol{\pi}(u)\,\mathrm{d}u = \mathbf{L}(t) &= \sum_{k=0}^{\infty} \boldsymbol{\pi}_0 P^k \int_0^t e^{-\alpha u} \frac{(\alpha u)^k}{k!}\,\mathrm{d}u \\ &= \sum_{k=0}^{\infty} \boldsymbol{\pi}_0 P^k \frac{1}{\alpha} \sum_{l=k+1}^{\infty} e^{-\alpha t} \frac{(\alpha t)^l}{l!} \\ &= \frac{1}{\alpha} \sum_{k=0}^{\infty} \boldsymbol{\pi}_0 P^k \left(1 - \sum_{l=0}^{k} e^{-\alpha t} \frac{(\alpha t)^l}{l!}\right). \end{aligned} \tag{5.11}$$

Both eqs. (5.10) and (5.11) on page 47 and on this page can be realized as

$$\mathbf{x} = \frac{1}{W} \left(\sum_{k=0}^{k_{\text{left}}-1} w_{\text{left}} \boldsymbol{\pi}_0 P^k + \sum_{k=k_{\text{left}}}^{k_{\text{right}}} w[k - k_{\text{left}}] \boldsymbol{\pi}_0 P^k \right), \tag{5.12}$$

where $\mathbf{x}$ is either $\boldsymbol{\pi}(t)$ or $\mathbf{L}(t)$, k_{left} and k_{right} are *trimming constants* selected based on the required precision, $\mathbf{w}$ is a vector of (possibly accumulated) Poisson weights and W is a scaling factor. The weight before the left cutoff w_{left} is 1 if the accumulated probability vector $\mathbf{L}(t)$ is calculated, 0 otherwise.

Eq. (5.12) is implemented by Algorithm 5.13. The algorithm performs *steady-state* detection in line 9 to avoid unneccessary work once the iteration vector $\mathbf{p}$ reaches the steady-state distribution $\boldsymbol{\pi}(\infty)$, i.e. $\mathbf{p} \approx \mathbf{p}P$. If the initial distribution $\boldsymbol{\pi}_0$ is not further needed or can be generated efficiently (as it is the case with a single initial state), the result vector $\mathbf{x}$ may share the same storing, resulting in a memory overhead of only two vectors $\mathbf{p}$ and $\mathbf{q}$.

The weights and trimming constants may be calculated by the famous algorithm of Fox and Glynn [5]. However, their algorithm is extremely complicated due to the limitations of single-precision floating-point arithmetic [7]. We implemented Burak's significantly simpler algorithm [4] in double precision instead (Algorithm 5.14 on page 50), which avoids underflow by a scaling factor $W \gg 1$.

5.2.2 TR-BDF2

A weakness of the uniformization algorithm is the poor tolerance of *stiff* Markov chains. The CTMC is called stiff if the $|\lambda_{\text{min}}| \ll |\lambda_{\text{max}}|$, where λ_{min} and λ_{max} are the nonzero eigenvalues of the infinitesimal generator matrix Q of minimum and maximum absolute value [12]. In other words, stiff Markov chains have behaviors on drastically different timescales, for example, clients are served frequently while failures happen infrequently.

Stiffness leads to very large values of α in line 2 of Algorithm 5.13, thus a large right cutoff k_{right} is required for computing the transient solution with sufficient accuracy.

Algorithm 5.13 Uniformization.

```
Input: infinitesimal generator $Q \in \mathbb{R}^{n \times n}$, initial probability vector $\boldsymbol{\pi}_0 \in \mathbb{R}^n$,
       truncation parameters $k_\text{left}, k_\text{right} \in \mathbb{N}$, weights $w_\text{left} \in \mathbb{R}$, $\mathbf{w} \in \mathbb{R}^{k_\text{right} - k_\text{left}}$,
       scaling constant $W \in \mathbb{R}$, tolerance $\tau > 0$
Output: instantenous or accumulated probability vector $\mathbf{x} \in \mathbb{R}^n$
allocate $\mathbf{x}, \mathbf{p}, \mathbf{q} \in \mathbb{R}^n$
$\alpha^{-1} \leftarrow 1/\max_i |a[i,i]|$
$\mathbf{p} \leftarrow \boldsymbol{\pi}_0$
if $w_\text{left} = 0$ then $\mathbf{x} \leftarrow \mathbf{0}$ else $\mathbf{x} \leftarrow w_\text{left} \cdot \mathbf{p}$      ▷ VectorScale
for $k \leftarrow 1$ to $k_\text{right}$ do
    $\mathbf{q} \leftarrow \mathbf{p}Q$      ▷ VectorMatrixMultiplyFromLeft
    $\mathbf{q} \leftarrow \alpha^{-1} \cdot \mathbf{q}$      ▷ In-place VectorScale
    $\mathbf{q} \leftarrow \mathbf{q} + \mathbf{p}$      ▷ In-place VectorAdd
    if $\|\mathbf{q} - \mathbf{p}\| \le \tau$ then
        $\mathbf{x} \leftarrow \mathbf{x} + \left(\sum_{l=k}^{k_\text{right}} w[l - k_\text{left}]\right) \cdot \mathbf{q}$      ▷ In-place VectorAdd
        break
    if $k < k_\text{left} \wedge w_\text{left} \ne 0$ then $\mathbf{x} \leftarrow \mathbf{x} + w_\text{left} \cdot \mathbf{q}$      ▷ In-place VectorAdd
    else if $k \ge k_\text{left}$ then $\mathbf{x} \leftarrow \mathbf{x} + w[k - k_\text{left}] \cdot \mathbf{q}$      ▷ In-place VectorAdd
    Swap the references to $\mathbf{p}$ and $\mathbf{q}$
$\mathbf{x} \leftarrow W^{-1} \cdot \mathbf{x}$      ▷ In-place VectorScale
return $\mathbf{x}$
```

Moreover, the slow stabilization results in taking many iterations before steady-state detection in line 9.

Some methods that can handle stiff CTMCs efficiently are stochastic complementation [9], which decouples the slow and fast behaviors of the system, and adaptive uniformization [10], which varies the uniformization rate α. Alternatively, an L-stable differential equation solver may be used to solve eq. (2.1) on page 4, such as TR-BDF2 [2, 12].

TR-BDF2 is an implicit integrator with alternating trapezoid rule (TR) steps

$$\boldsymbol{\pi}_{k+\gamma}(2I + \gamma h_k Q) = 2\boldsymbol{\pi}_k + \gamma h_k \boldsymbol{\pi}_k Q$$

and second order backward difference steps

$$\boldsymbol{\pi}_{k+1}[(2-\gamma)I - (1-\gamma)h_k Q] = \frac{1}{\gamma}\boldsymbol{\pi}_{k+\gamma} - \frac{(1-\gamma)^2}{\gamma}\boldsymbol{\pi}_k,$$

which advance the time together by a step of size h_k. The constant $0 < \gamma < 1$ sets the breakpoint between the two steps. We set it to $\gamma = 2 - \sqrt{2} \approx 0.59$ following the recommendation of Bank et al. [2].

Algorithm 5.14 Burak's algorithm for calculating the Poisson weights.

```
Input: Poisson rate $\lambda = \alpha t$, tolerance $\tau > 10^{-50}$
Output: truncation parameters $k_{\text{left}}, k_{\text{right}} \in \mathbb{N}$, weights $\mathbf{w} \in \mathbb{R}^{k_{\text{right}} - k_{\text{left}}}$, scaling
        constant $W \in \mathbb{R}$

Calculate weights with high precision
$M_w \leftarrow 30, M_a \leftarrow 44, M_s \leftarrow 21$   // Constants determine cutoff estimation accuracy
$m \leftarrow \lfloor \lambda \rfloor, tSize \leftarrow \lfloor M_w \sqrt{\lambda} + M_a \rfloor, tStart \leftarrow \max\{m + M_s - \lfloor tSize/2 \rfloor, 0\}$
allocate tWeights $\in \mathbb{R}^{tSize}$
$tWeights[m - tStart] \leftarrow 2^{176}$
for $j \leftarrow m - tStart$ downto $1$ do
    $tWeights[j-1] = (j + tStart)\, tWeights[j]/\lambda$
for $j \leftarrow m - tStart + 1$ to $tSize$ do
    $tWeights[j+1] = \lambda\, tWeights[j]/(j + tStart)$

Determine normalization constant and cutoff points
$W \leftarrow 0$
for $j \leftarrow 0$ to $m - tStart - 1$ do
    $W \leftarrow W + tWeights[j]$
$sum1 \leftarrow 0$   // Avoid adding small numbers to larger numbers
for $j \leftarrow tSize - 1$ downto $m - tStart$ do
    $sum1 \leftarrow sum1 + tWeights[j]$
$W \leftarrow W + sum1, threshold \leftarrow W\tau/2, cdf \leftarrow 0, i \leftarrow 0$
while $cdf < threshold$ do
    $cdf \leftarrow cdf + tWeights[i]$
    $i \leftarrow i + 1$
$k_{\text{left}} \leftarrow tStart + i, cdf \leftarrow 0, i \leftarrow tSize - 1$
while $cdf < threshold$ do
    $cdf \leftarrow cdf + tWeights[i]$
    $i \leftarrow i - 1$
$k_{\text{right}} \leftarrow tStart + i$

Copy weights between cutoff points
allocate $\mathbf{w} \in \mathbb{R}^{k_{\text{right}} - k_{\text{left}}}$
for $j \leftarrow k_{\text{left}}$ to $k_{\text{right}}$ do
    $w[j - k_{\text{left}}] \leftarrow tWeights[j - tStart]$
return $k_{\text{left}}, k_{\text{right}}, \mathbf{w}, W$
```

As a guess for the initial step size h_0, we chose the uniformization rate of Q. The kth step size $h_k > 0$, including the 0th one, is selected such that the local error estimate

$$LTE_{k+1} = \left\| 2\frac{-3\gamma^4 + 4\gamma - 2}{24 - 12\gamma} h_k \left[-\frac{1}{\gamma}\boldsymbol{\pi}_k + \frac{1}{\gamma(1-\gamma)}\boldsymbol{\pi}_{k+\gamma} - \frac{1}{1-\gamma}\boldsymbol{\pi}_{k+1} \right] \right\| \tag{5.13}$$

is bounded by the local error tolerance

$$LTE_{k+1} \le \left(\frac{\tau - \sum_{i=0}^{k} LTE_i}{t - \sum_{i=0}^{k} k_i} \right) h_{k+1}.$$

This Local Error per Unit Step (LEPUS) error control "produces excellent results for many problems", but is usually costly [12]. Moreover, the accumulated error at the end of integration may be larger than the prescribed tolerance τ, since eq. (5.13) is only an approximation of the true error.

An implementation of TR-BDF2 based on the pseudocode of A. L. Reibman and Trivedi [12] is shown in Algorithm 5.15.

In lines 10 and 13 any linear equation solver from Section 5.1 on page 30 may be used except power iteration, since the matrices, in general, do not have strictly negative diagonals. Due to the way the matrices, which are linear combinations of I and Q, are passed to the inner solvers, our TR-BDF2 integrator is currently limited to Q matrices which are not in block form.

The vectors $\boldsymbol{\pi}_0, \boldsymbol{\pi}_k$ and $\boldsymbol{\pi}_{k+\gamma}, \mathbf{d}_{k+1}$ may share storage, respectively, therefore only 4 state-space sized vectors are required in addition to the initial distribution $\boldsymbol{\pi}_0$.

The most computationally intensive part is the solution of two linear equation per every attempted step, which may make TR-BDF2 extremely slow. However, its performance does *not* depend on the stiffness of the Markov chain, which may make it better suited to stiff CTMCs than uniformization [12].

5.3 Mean time to first failure

In MTFF calculation (Section 2.1.3 on page 7), quantities of the forms

$$MTFF = -\underbrace{\boldsymbol{\pi}_U Q_{UU}^{-1}}_{\gamma} \mathbf{1}^{\mathrm{T}}, \quad \mathbb{P}(X(TFF_{+0}) = y) = -\underbrace{\boldsymbol{\pi}_U Q_{UU}^{-1}}_{\gamma} \mathbf{q}_{UD'}^{\mathrm{T}} \tag{2.7, 2.8 revisited}$$

are computed, where U, D, D' are the set of operations states, failure states and a specific failure mode $D' \subsetneq D$, respectively.

The vector $\boldsymbol{\gamma} \in \mathbb{R}^{|U|}$ is the solution of the linear equation

$$\boldsymbol{\gamma} Q_{UU} = \boldsymbol{\pi}_U \tag{5.14}$$

Algorithm 5.15 TR-BDF2 for transient analysis.

```
Input: infinitesimal generator $Q \in \mathbb{R}^{n \times n}$, initial distribution $\boldsymbol{\pi}_0$,
mission time $t > 0$, tolerance $\tau > 0$
Output: transient distribution $\boldsymbol{\pi}(t)$
allocate $\boldsymbol{\pi}_k, \boldsymbol{\pi}_{k+\gamma}, \boldsymbol{\pi}_{k+1}, \mathbf{d}_k, \mathbf{d}_{k+1}, \mathbf{y} \in \mathbb{R}^n$
$maxIncrease \leftarrow 10, leastDecrease \leftarrow 0.9$
$timeLeft \leftarrow t, h \leftarrow 1/\max_i |q[i,i]|, \gamma \leftarrow 2 - \sqrt{2}, C \leftarrow \left|\frac{-3\gamma^4+4\gamma-2}{24-12\gamma}\right|, errorSum \leftarrow 0$
$\boldsymbol{\pi}_k \leftarrow \boldsymbol{\pi}_0$, $\mathbf{d}_k \leftarrow \boldsymbol{\pi}_k Q$    ▷ VectorMatrixMultiplyFromLeft
while $timeLeft > 0$ do
    $stepFailed \leftarrow$ false, $h \leftarrow \min\{h, timeLeft\}$
    while true do
        TR step
        $\mathbf{y} \leftarrow 2 \cdot \boldsymbol{\pi}_k$    ▷ VectorScale
        $\mathbf{y} \leftarrow \mathbf{y} + \gamma h \cdot \mathbf{d}_k$    ▷ In-place VectorAdd
        Solve $\boldsymbol{\pi}_{k+\gamma}(2I + -\gamma hQ) = \mathbf{y}$ for $\boldsymbol{\pi}_{k+\gamma}$ with initial guess $\boldsymbol{\pi}_k$
        BDF2 step
        $\mathbf{y} \leftarrow -\frac{(1-\gamma)^2}{\gamma} \cdot \boldsymbol{\pi}_k$    ▷ VectorScale
        $\mathbf{y} \leftarrow \frac{1}{\gamma} \cdot \boldsymbol{\pi}_{k+\gamma}$    ▷ In-place VectorScale
        Solve $\boldsymbol{\pi}_{k+1}((2-\gamma)I + (\gamma-1)hQ) = \mathbf{y}$ for $\boldsymbol{\pi}_{k+1}$ with initial guess $\boldsymbol{\pi}_{k+\gamma}$
        Error control and step size estimation
        $\mathbf{y} \leftarrow -\frac{1}{\gamma}\mathbf{d}_k$    ▷ VectorScale
        $\mathbf{y} \leftarrow \mathbf{y} + \frac{1}{\gamma(1-\gamma)}\boldsymbol{\pi}_{k+\gamma} Q$    ▷ VectorMatrixAccumulateMultiplyFromLeft
        $\mathbf{d}_{k+1} \leftarrow \boldsymbol{\pi}_{k+1} Q$    ▷ VectorMatrixMultipleFromLeft
        $\mathbf{y} \leftarrow \mathbf{y} + \left(-\frac{1}{1-\gamma}\right)\mathbf{d}_{k+1}$    ▷ In-place VectorAdd
        $LTE \leftarrow 2Ch\|\mathbf{y}\|, localTol \leftarrow (\tau - errorSum)/timeLeft \cdot h$
        if $LTE < localTol$ then    // Successful step
            $timeLeft \leftarrow timeLeft - h, errorSum \leftarrow errorSum + LTE$
            // Do not try to increase h after a failed step
            if $\neg stepFailed$ then $h \leftarrow h \cdot \min\{maxIncrease, \sqrt[3]{localTol/LTE}\}$
            break
        $stepFailed \leftarrow$ true, $h \leftarrow h \cdot \min\{leastDecrease, \sqrt[3]{localTol/LTE}\}$
    Swap the references to $\boldsymbol{\pi}_k, \boldsymbol{\pi}_{k+1}$ and $\mathbf{d}_k, \mathbf{d}_{\mathbf{k+1}}$
return $\boldsymbol{\pi}_k$
```

and may be obtained by any linear equation solver.

The sets $U, D = D_1 \cup D_2 \cup \cdots$ are constructed by the evaluation of CTL expressions. If the failure mode D_i is described by φ_i, then the sets D and U are described by CTL formulas $\varphi_D = \neg\mathbf{AX}\,\text{true} \vee \varphi_1 \vee \varphi_2 \vee \cdots$ and $\varphi_U = \neg\varphi_D$, where the deadlock condition $\neg\mathbf{AX}\,\text{true}$ is added to make (5.14) irreducible.

After the set U is generated symbolically, the matrix Q_{UU} may be decomposed in the same way as the whole state space S. Thus, the vector-matrix operations required for solving (5.14) can be executed as in steady-state analysis.

References

[1] Kensuke Aihara, Kuniyoshi Abe, and Emiko Ishiwata. “A variant of IDRstab with reliable update strategies for solving sparse linear systems”. In: *Journal of Computational and Applied Mathematics* 259 (2014), pp. 244–258.

[2] Randolph E. Bank, William M. Coughran Jr., Wolfgang Fichtner, Eric Grosse, Donald J. Rose, and R. Kent Smith. “Transient Simulation of Silicon Devices and Circuits”. In: *IEEE Trans. on CAD of Integrated Circuits and Systems* 4.4 (1985), pp. 436–451. DOI: 10.1109/TCAD.1985.1270142.

[3] Richard Barrett, Michael W Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*. Vol. 43. Siam, 1994.

[4] Maciej Burak. “Multi-step Uniformization with Steady-State Detection in Nonstationary M/M/s Queuing Systems”. In: *CoRR* abs/1410.0804 (2014). URL: http://arxiv.org/abs/1410.0804.

[5] Bennett L. Fox and Peter W. Glynn. “Computing Poisson Probabilities”. In: *Commun. ACM* 31.4 (1988), pp. 440–445. DOI: 10.1145/42404.42409.

[6] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1997. ISBN: 9781611970937. URL: https://books.google.hu/books?id=IX9rrFe1YLQC.

[7] David N Jansen. “Understanding Fox and Glynn’s “Computing Poisson probabilities”. In: (2011).

[8] Amy Nicole Langville and William J. Stewart. “Testing the Nearest Kronecker Product Preconditioner on Markov Chains and Stochastic Automata Networks”. In: *INFORMS Journal on Computing* 16.3 (2004), pp. 300–315. DOI: 10.1287/ijoc.1030.0041.

[9] Carl D Meyer. “Stochastic complementation, uncoupling Markov chains, and the theory of nearly reducible systems”. In: *SIAM review* 31.2 (1989), pp. 240–272.

[10] Aad PA van Moorsel and William H Sanders. “Adaptive uniformization”. In: *Stochastic Models* 10.3 (1994), pp. 619–647.

[11] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[12] Andrew L. Reibman and Kishor S. Trivedi. “Numerical transient analysis of markov models”. In: *Computers & OR* 15.1 (1988), pp. 19–36. DOI: 10.1016/0305-0548(88)90026-3.

[13] Andrew Reibman, Roger Smith, and Kishor Trivedi. “Markov and Markov reward model transient analysis: An overview of numerical approaches”. In: *European Journal of Operational Research* 40.2 (1989), pp. 257–267.

[14] Youcef Saad and Martin H Schultz. “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems”. In: *SIAM Journal on scientific and statistical computing* 7.3 (1986), pp. 856–869.

[15] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[16] Valeria Simoncini and Daniel B Szyld. “Interpreting IDR as a Petrov–Galerkin method”. In: *SIAM Journal on Scientific Computing* 32.4 (2010), pp. 1898–1912.

[17] Gerard LG Sleijpen and Diederik R Fokkema. “BiCGstab (l) for linear equations involving unsymmetric matrices with complex spectrum”. In: *Electronic Transactions on Numerical Analysis* 1.11 (1993), p. 2000.

[18] Gerard LG Sleijpen and Martin B Van Gijzen. “Exploiting BiCGstab (ℓ) strategies to induce dimension reduction”. In: *SIAM journal on scientific computing* 32.5 (2010), pp. 2687–2709.

[19] Peter Sonneveld. “CGS, a fast Lanczos-type solver for nonsymmetric linear systems”. In: *SIAM journal on scientific and statistical computing* 10.1 (1989), pp. 36–52.

[20] Peter Sonneveld and Martin B van Gijzen. “IDR (s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations”. In: *SIAM Journal on Scientific Computing* 31.2 (2008), pp. 1035–1062.

[21] William J Stewart. *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton University Press, 2009.

[22] Masaaki Tanio and Masaaki Sugihara. “GBi-CGSTAB (s, L): IDR (s) with higher-order stabilization polynomials”. In: *Journal of computational and applied mathematics* 235.3 (2010), pp. 765–784.

[23] Henk A Van der Vorst. “Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems”. In: *SIAM Journal on scientific and Statistical Computing* 13.2 (1992), pp. 631–644.