Back in Time. Fast.
Accelerated Time Iterations

Pablo Winant

31/10/2025

Abstract

We present two complementary algorithms to solve nonlinear rational expectations models characterized by first order conditions: an accelerated time-iteration method and a Newton–Krylov solver. Both approaches exploit an explicit construction of the derivative operator (and the model Jacobian) and achieve quadratic convergence near the solution, yielding large computational gains over standard time iteration. We show how to apply these linear operators without forming dense matrices and invert the resulting systems efficiently using truncated Neumann series or GMRES. On three benchmark models (consumption–savings, RBC, and a two-country model), the two methods produce the same solution with substantially reduced runtimes.

Keywords: Time Iterations, Acceleration, Quadratic Convergence, Newton–Krylov, Iterative Solvers

1 Introduction

We present two complementary algorithms to solve a wide class of nonlinear rational expectations models characterized by first-order conditions: an accelerated time-iteration method and a Newton–Krylov solver.[1]

This paper is motivated by the growing interest in the economics profession to investigate the robustness of nonlinear medium-scale rational equilibrium models, when the economy can be drawn away from the steady state or when there are important nonlinearities such as an occasionally binding constraint or a zero lower bound (for instance Fernández-Villaverde et al. (2015) and Coeurdacier, Rey, and Winant (2020)).

In general, these models do not derive from the problem of one single optimizing agent, hence methods based on Bellman equations like value iteration or Howard policy iterations (see Rust (1996) for a review) can't be used. Instead we intend to solve any model which can be characterized by a set of transition and equilibrium equations. The latter include Euler equations, asset prices, or other nonlinear relations.

This generic class of model is typically solved by the time-iteration algorithm (Coleman, 1990; Deaton and Laroque, 1992). The time iteration, sometimes referred to as Coleman iteration, consists in finding the optimal decision rule today, knowing the decision rule tomorrow. Starting from an initial guess and applying the operator many times, i.e., going back in time, yields the solution to the model.

Many methods have been proposed to accelerate the solution for this family of problems. Among them, the parameterized expectations (den Haan and Marcet, 1990), the endogenous grid point methods (Carroll, 2006), or the precomputation of expectations (K. L. Judd, L. Maliar, S. Maliar, and Tsener, 2017) reduce the cost of each iteration but only work when the first-order conditions can be recast in a specific form. By contrast, our method only requires the asymptotic convergence of the time iterations, a desirable property of any well-defined model.

The literature has also discussed several ways to increase the precision of the approximate decision rules or to reduce the number of parameters. Splines, Chebyshev, and complete polynomials are discussed in K. L. Judd, 1998; Smolyak polynomials in K. L. Judd, L. Maliar, S. Maliar, and Valero (2014). All those linear interpolation schemes are compatible with our algorithm. Other attempts to reduce the dimensionality of rational expectation models have resorted to adaptive grids (Brumm and Scheidegger, 2017), ergodic sets (S.

[1]Part of the research was conducted while the author was at the Bank of England. The author would like to thank Basile Grassi, Michel Juillard, Michael Kumhof, Lilia Maliar, Serguei Maliar, Riccardo Masolo, Wanda Mimra, Michael Reiter, John Stachurski and Anastasia Zhutova for valuable help or feedback at different stages of this project.

Maliar, L. Maliar, and K. Judd, 2011) or neural networks (L. Maliar, S. Maliar, and Winant, 2021; Azinovic, Gaegauf, and Scheidegger, 2022).

The method we propose accelerates each time iteration by using its derivative operator to produce an estimate of the fixed point.[2] While the procedure produces exactly the same solution as the common time-iteration algorithm, it converges at a *quadratic* rate instead of a *geometric* one and completes much faster in practical applications.

One key contribution of this paper resides in showing how to construct an explicit representation of the derivative operator. Since this linear operator is not easily represented as a matrix, we need to resort to iterative methods to solve any linear system that involves it. Many iterative solvers have been developed for this task, including GMRES (Saad and Schultz, 1986), but in our experiments we show that a very simple recursive calculation performs almost as well, with a lower memory footprint. We also show how the Jacobian of the full model can be computed with very similar steps, allowing for a direct solution of the model as a single nonlinear system.

Section 2 introduces the general form of the models that we solve. Section 3 sketches out our main algorithm, while Section 4 describes an efficient way to represent the model Jacobians that are needed to implement it. Section 5 discusses several options to invert the Jacobian and draws an explicit connection with iterative solvers. Section 6 discusses the implementation of a simple Newton-Krylov solver. Section 7 compares and explains computation times on three different models, that are detailed separately in Section A. Section 8 concludes.

2 General Setup

Consider a state space $\mathcal{S} \subset \mathbb{R}^{n_s}$ and an unbounded space of continuous controls $\mathcal{X} = \mathbb{R}^{n_x}$.

In any state $s \in \mathcal{S}$, the optimal controls $x \in \mathcal{X}$ are characterized by

$$\mathbb{E}\big[f(s, x, s', x')\big] = 0 \tag{1}$$

where f represents a set of n_x equations, where future states s' are drawn after distribution $\tau^m(s, x)$ and where future controls x' are taken according to an initially unknown decision rule $\phi(s') = x'$.

The specification $(\mathcal{S}, \mathcal{X}, f, \tau^m)$ can represent models with both discrete and continuous states, depending on the definition of $\mathcal{S}$. It contains a wide class of economic models, the main limitation being that we require controls to be continuously valued, so as to be naturally characterized by Euler equation (1).

2.1 Discretization

To solve the model approximately, we discretize the state-space into a finite vector of N states $\overline{\boldsymbol{s}} = \big(\overline{\boldsymbol{s}}^n\big)_{n\in\{1,\dots,N\}} \in \mathcal{S}^N$.

To a vector of corresponding controls[3] $\overrightarrow{\boldsymbol{x}} = (\overrightarrow{\boldsymbol{x}}^n)_{n\in\{1\dots N\}} \in \mathcal{X}^N$ we associate an interpolation operator $\mathcal{J}$ in order to define a decision rule over the whole state-space: $s \in \mathcal{S} \mapsto x = \mathcal{J}(s; \overrightarrow{\boldsymbol{x}}) \in \mathcal{X}$. Crucially, we assume that $\mathcal{J}$ is *linear in the data* $\overrightarrow{\boldsymbol{x}}$ to be interpolated. This assumption is satisfied by many common interpolation methods including splines of any order, all spectral polynomial methods (complete, Chebyshev, Smolyak) and radial bases.[4]

[2] In some sense our algorithm is to Euler equations what Howard policy iteration is for Bellman problems. Indeed, Howard improvement consists in a Bellman update (our time iteration) followed by a policy evaluation (our acceleration). We know since Puterman and Brumelle (1979) that the latter step can equivalently be expressed as a function of the (sub-)derivative of the Bellman update. Like our method, Howard policy iteration is thus an accelerated fixed-point algorithm. In particular it also converges quadratically.

[3] In this paper, we use lowercase (like s, x) to denote particular values for states and controls. *Vectors* enumerating values on grid are in bold: we use $\overrightarrow{\boldsymbol{x}}$ for today, $\overrightarrow{\boldsymbol{y}}$ for tomorrow and $\overline{\boldsymbol{x}}$ for the steady-state (as in $\overline{\boldsymbol{s}}$ and $\overline{\boldsymbol{x}}$). Coordinates of these vectors are denoted with a superscript: for instance $\overrightarrow{\boldsymbol{x}}^n$ is a control today at grid point $\overline{\boldsymbol{s}}^n$. Conventionally, we use subscripts to index sequences in this vector space, as in $(\overrightarrow{\boldsymbol{x}}_k)_{k\geq 0}$.

[4] For these methods there exists a base of (possibly nonlinear) functions $(\mathcal{B}_n)_{n\in\{1\dots N\}}$, and a set of coefficients $(c_n)_{n\in\{1\dots N\}}$ such that $\mathcal{J}(s; \overrightarrow{\boldsymbol{x}}) = \sum_n c_n \mathcal{B}_n(s)$. These coefficients are obtained by solving in c_n the system $\overrightarrow{\boldsymbol{x}}^n = \sum_n c_n \mathcal{B}(\overline{\boldsymbol{s}}^n)$. Since this system is linear in c_n there exists a set of linear functions $(m_n)_{n\in\{1\dots N\}}$ such that $c_n = m_n \cdot \overrightarrow{\boldsymbol{x}}$ and as a result we have the formula $\mathcal{J}(s; \overrightarrow{\boldsymbol{x}}) = \sum_n (m_n \cdot \overrightarrow{\boldsymbol{x}})\mathcal{B}_n(s)$ which is clearly *linear in the data* $\overrightarrow{\boldsymbol{x}}$. By contrast, wavelets or neural networks do not satisfy this property.

We also approximate the transition distribution $\tau^m(s,x)$ by a finite distribution $\tau(s,x)$ materialized by enumerating the K future states $(s'_i(s,x))_{i\in\{1\dots K\}}$ occurring with probabilities $(w_i(s,x))_{i\in\{1\dots K\}}$.

This leads to the *computable* residual function

$$F(s,x;\overrightarrow{\boldsymbol{y}}) = \sum_{(w,s')\in\tau(s,x)} w\, f\big(s,x,s',\mathcal{J}(s';\overrightarrow{\boldsymbol{y}})\big) \tag{2}$$

where tomorrow's decision rule $\mathcal{J}(\,\cdot\,;\overrightarrow{\boldsymbol{y}})$ is characterized by the vector of values $\overrightarrow{\boldsymbol{y}}$ on the grid $\overline{\boldsymbol{s}}$.

Finally, for any vector of controls today $\overrightarrow{\boldsymbol{x}}$ we define the vector of residuals as

$$\boldsymbol{F}\big(\overrightarrow{\boldsymbol{x}},\overrightarrow{\boldsymbol{y}}\big) = \Big(F\big(\overline{\boldsymbol{s}}^n,\overrightarrow{\boldsymbol{x}}^n;\overrightarrow{\boldsymbol{y}}\big)\Big)_{n\in\{1\dots N\}} \tag{3}$$

where we leave implicit the dependence of $\boldsymbol{F}$ on $\overline{\boldsymbol{s}}$ since the latter remains constant. Equation (3) represents the discretized model to be solved numerically.

Time invariant solution $\overline{\boldsymbol{x}} \in \mathcal{X}^N$ to the discretized model must satisfy:

$$\boldsymbol{G}\big(\overline{\boldsymbol{x}}\big) = \boldsymbol{F}\big(\overline{\boldsymbol{x}},\overline{\boldsymbol{x}}\big) = 0. \tag{4}$$

3 Accelerated Time Iteration

We now outline two simple algorithms to solve any model that can be represented as in equation (3).

3.1 Time Iteration

Let's define the time iteration operator as the function $\mathcal{T} : \mathcal{X}^N \to \mathcal{X}^N$ such that, for any given $\overrightarrow{\boldsymbol{x}}$,

$$F\big(\mathcal{T}(\overrightarrow{\boldsymbol{x}}),\overrightarrow{\boldsymbol{x}}\big) = 0. \tag{5}$$

The time iteration algorithm consists in computing the iterates

$$\overrightarrow{\boldsymbol{x}}_{k+1} = \mathcal{T}\big(\overrightarrow{\boldsymbol{x}}_k\big) \tag{6}$$

starting from an initial guess $\overrightarrow{\boldsymbol{x}}_0$.

Convergence is guaranteed if $\overrightarrow{\boldsymbol{x}}_0$ is close enough to $\overline{\boldsymbol{x}}$ under the condition

$$\rho\big(\mathcal{T}'(\overline{\boldsymbol{x}})\big) < 1$$

where $\mathcal{T}'(\overline{\boldsymbol{x}})$ is the Fréchet derivative of $\mathcal{T}$ at $\overline{\boldsymbol{x}}$ and function $\rho(\,\cdot\,)$ denotes the spectral radius.

Convergence is asymptotically *geometric*, with a decay rate faster than $\rho(\mathcal{T}') + \epsilon$ for any $\epsilon > 0$.

3.2 Accelerated Time Iteration

Building on $\mathcal{T}$ and $\mathcal{T}'$, we propose a simple acceleration method.

Assuming one time-iteration step $\tilde{\boldsymbol{x}}_{k+1} = \mathcal{T}(\overrightarrow{\boldsymbol{x}}_k)$ has been computed, we can build a *linear* approximation of the time operator $\mathcal{T}$ around $\overrightarrow{\boldsymbol{x}}_k$ as[5]

$$\mathcal{T}_k(\boldsymbol{x}) = \overrightarrow{\boldsymbol{x}}_{k+1} + \mathcal{T}'\big(\overrightarrow{\boldsymbol{x}}_k\big)\cdot\big(\boldsymbol{x} - \overrightarrow{\boldsymbol{x}}_k\big).$$

Then a fixed point of $\mathcal{T}$ can be approximated by solving for $\mathcal{T}_k(\overrightarrow{\boldsymbol{x}}) = \overrightarrow{\boldsymbol{x}}$ in $\overrightarrow{\boldsymbol{x}}$ which yields a new guess:

$$\begin{aligned}\overrightarrow{\boldsymbol{x}}_{k+1} &= \mathcal{A}\big(\tilde{\boldsymbol{x}}_{k+1},\overrightarrow{\boldsymbol{x}}_k\big)\\ &= \overrightarrow{\boldsymbol{x}}_k + \big(I - \mathcal{T}'(\overrightarrow{\boldsymbol{x}}_k)\big)^{-1}\cdot\big(\tilde{\boldsymbol{x}}_{k+1} - \overrightarrow{\boldsymbol{x}}_k\big).\end{aligned} \tag{7}$$

[5]Throughout the paper we use the notation $L\cdot u$ to denote the application of a linear operator L to a vector u. For instance if A, B and M are all square matrices in $\mathbb{R}^{n\times n}$, the operator $L(M) = AM - MB$ is clearly linear in M but is not represented by a matrix in $\mathbb{R}^{n\times n}$. We then write $L\cdot M$ instead of $L(M)$. The dot is meant to remind the reader that this is not a simple matrix multiplication.

Note that this expression is well defined, since, by continuity $\rho(\mathcal{T}'(\overrightarrow{\boldsymbol{x}}_k)) < 1$ close enough to the solution.

The accelerated time iteration algorithm consists in computing the iterates:

$$\overrightarrow{\boldsymbol{x}}_{k+1} = \mathcal{A}\big(\mathcal{T}(\overrightarrow{\boldsymbol{x}}_k), \overrightarrow{\boldsymbol{x}}_k\big). \tag{8}$$

The sequence of accelerated time iterations also converges to $\overline{\boldsymbol{x}}$ when the initial guess $\overrightarrow{\boldsymbol{x}}_0$ is close enough to the steady-state.

Furthermore, the convergence is *quadratic*.[6]

4 Computing the derivatives

The two algorithms we have described in Section 3 require the calculation of operators $\mathcal{T}(\overrightarrow{\boldsymbol{x}}_k)$ and $\mathcal{T}'(\overrightarrow{\boldsymbol{x}}_k)$. In this section we describe how they can be obtained from a suitable representation of the derivative of F with respect to its first (resp. second) argument F'_A (resp. F'_B).

First, $\mathcal{T}(\overrightarrow{\boldsymbol{x}}_k)$ can be obtained by applying a Newton solver to the function $u \to F(u, \overrightarrow{\boldsymbol{x}}_k)$ that uses derivative F'_A.

Then, once a solution $\overrightarrow{\boldsymbol{x}}_{k+1} = \mathcal{T}(\overrightarrow{\boldsymbol{x}}_k)$ is found that satisfies $F(\mathcal{T}(\overrightarrow{\boldsymbol{x}}_k), \overrightarrow{\boldsymbol{x}}_k) = 0$ we can differentiate equation (5)

$$F'_A\big(\overrightarrow{\boldsymbol{x}}_{k+1}, \overrightarrow{\boldsymbol{x}}_k\big)\mathcal{T}'(\overrightarrow{\boldsymbol{x}}_k) + F'_B\big(\overrightarrow{\boldsymbol{x}}_{k+1}, \overrightarrow{\boldsymbol{x}}_k\big) = 0$$

to get

$$\mathcal{T}'(\overrightarrow{\boldsymbol{x}}_k) = -F'_A\big(\overrightarrow{\boldsymbol{x}}_{k+1}, \overrightarrow{\boldsymbol{x}}_k\big)^{-1} F'_B\big(\overrightarrow{\boldsymbol{x}}_{k+1}, \overrightarrow{\boldsymbol{x}}_k\big). \tag{9}$$

In the next subsection we show how to compute operators $F'_A(\,\cdot\,,\,\cdot\,)$ and $F'_B(\,\cdot\,,\,\cdot\,)$ and how to combine them to compute equation (9).

4.1 Computing F'_A

From the definition of F in equation (3), it is apparent that the n-th component of F only depends on the n-th component of $\overrightarrow{\boldsymbol{x}}$.

Hence F'_A is a matrix with a block diagonal structure where the diagonal elements are given by:

$$\begin{aligned}\boldsymbol{F}'_A(\overrightarrow{\boldsymbol{x}}, \overrightarrow{\boldsymbol{y}})_{n,n} &= F'_x\big(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n; \overrightarrow{\boldsymbol{y}}\big) \\ &= \sum_{(w,s') \in \tau(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n)} w\, f'_x\big(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n, s', \mathcal{J}(s'; \overrightarrow{\boldsymbol{y}})\big).\end{aligned} \tag{10}$$

4.2 Computing $F'_B \cdot \overrightarrow{\boldsymbol{u}}$

As shown in Figure 1, the matrix representation of F'_B does not have a very simple structure. It is however possible to efficiently compute $F'_B \cdot \overrightarrow{\boldsymbol{u}}$ for many vectors $\overrightarrow{\boldsymbol{u}}$ efficiently.

Since the function $\mathcal{J}$ used to interpolate future controls as $\mathcal{J}(s; \overrightarrow{\boldsymbol{y}})$ is linear in $\overrightarrow{\boldsymbol{y}}$ we can write for any $\overrightarrow{\boldsymbol{u}}$,

$$\mathcal{J}'_{\boldsymbol{x}}(s; \overrightarrow{\boldsymbol{y}}) \cdot \overrightarrow{\boldsymbol{u}} = \mathcal{J}(s; \overrightarrow{\boldsymbol{u}}).$$

This insight allows us to differentiate equation (2) w.r.t. x as follows:

$$F'_B(s, x; \overrightarrow{\boldsymbol{y}}) \cdot \overrightarrow{\boldsymbol{u}} = \sum_{(w,s') \in \tau(s,x)} w\, f'_{x'}\big(s, x, s', \mathcal{J}(s'; \overrightarrow{\boldsymbol{y}})\big)\mathcal{J}(s'; \overrightarrow{\boldsymbol{u}}). \tag{11}$$

Finally, we get the expression:

$$\boldsymbol{F}'_B(\overrightarrow{\boldsymbol{x}}, \overrightarrow{\boldsymbol{y}}) \cdot \overrightarrow{\boldsymbol{u}} = \Big(F'_B\big(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n; \overrightarrow{\boldsymbol{y}}\big) \cdot \overrightarrow{\boldsymbol{u}}\Big)_{n \in \{1 \dots N\}}.$$

[6] This results from observing that equation (8) produces the same iterates as a Newton-Raphson scheme applied to $u \to T(u) - u$. This is shown in Section B.3 of the appendix.

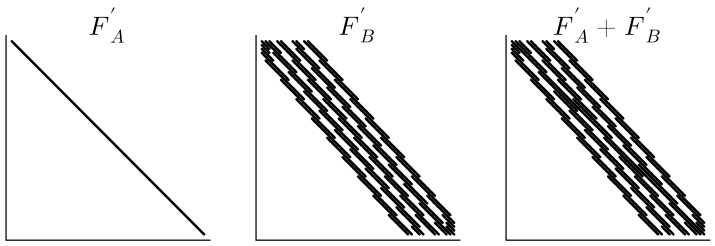

Figure 1: Structure of the jacobian for the RBC model

4.3 Computing $(F_A')^{-1}F_B' \cdot \boldsymbol{u}$

We now want to compute the vector

$$\begin{aligned} L \cdot \overrightarrow{\boldsymbol{u}} &= F_A'(\overrightarrow{\boldsymbol{x}}, \overrightarrow{\boldsymbol{y}})^{-1} F_B'(\overrightarrow{\boldsymbol{x}}, \overrightarrow{\boldsymbol{y}}) \cdot \overrightarrow{\boldsymbol{u}} \\ &= \left(F_A'\big(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n; \overrightarrow{\boldsymbol{y}}\big)^{-1} F_B'\big(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n; \overrightarrow{\boldsymbol{y}}\big) \cdot \overrightarrow{\boldsymbol{u}} \right)_{n \in \{1 \dots N\}} \end{aligned}$$

for any $\overrightarrow{\boldsymbol{u}}$. This defines the linear operator $L(\overrightarrow{\boldsymbol{x}}, \overrightarrow{\boldsymbol{y}})$.

Because F_A' has a block diagonal structure, we can compute for any grid point $\overline{\boldsymbol{s}}^n$ and choice $\overrightarrow{\boldsymbol{x}}^n$:

$$\begin{aligned} & F_A'\big(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n; \overrightarrow{\boldsymbol{y}}\big)^{-1} F_B'\big(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n; \overrightarrow{\boldsymbol{y}}\big) \cdot \overrightarrow{\boldsymbol{u}} \\ &= \sum_{(w,s') \in \tau(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n)} \underbrace{w\, f_x'\big(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n, s', x'\big)^{-1} f_{x'}'\big(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n, s', x'\big)}_{D(\overline{\boldsymbol{s}}^n, s')} \mathcal{J}(s'; \overrightarrow{\boldsymbol{u}}). \end{aligned} \tag{12}$$

For the algorithms we consider, repeated evaluations of $L \cdot \overrightarrow{\boldsymbol{u}}$ at various values of $\overrightarrow{\boldsymbol{u}}$ take most of the time. It is then profitable to find a way to increase their efficiency. For that purpose, we can precompute all the terms appearing in equation (12) that is

$$\left(s', \big(D(\overline{\boldsymbol{s}}^n, s')\big)_{(w',s') \in \tau(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n)} \right)_{n \in \{1 \dots N\}}$$

and store the result in two matrices

$$\begin{aligned} S &= (s_{n,k})_{n \in \{1 \dots N\}, k \in \{1 \dots K\}} \\ D &= \big(D(s_n, s_{n,k})\big)_{n \in \{1 \dots N\}, k \in \{1 \dots K\}} \end{aligned}$$

whose elements are n_s vectors and $n_x \times n_x$ matrices respectively.

Then to apply L on a vector $\overrightarrow{\boldsymbol{u}}$ we just have to compute:

$$L \cdot \overrightarrow{\boldsymbol{u}} = \sum_n \sum_k D_{n,k}\, \mathcal{J}(S_{n,k}; \overrightarrow{\boldsymbol{u}}). \tag{13}$$

Note that linear operator $L(\overrightarrow{\boldsymbol{x}}, \overrightarrow{\boldsymbol{y}})$ can be computed at any two $\overrightarrow{\boldsymbol{x}}$, $\overrightarrow{\boldsymbol{y}}$. As a special case we have $\mathcal{T}'(\overrightarrow{\boldsymbol{x}}) = L(\mathcal{T}(\overrightarrow{\boldsymbol{x}}), \overrightarrow{\boldsymbol{x}})$ which is what is required by the accelerated time-iterations. We will show in Section 6 that we can also use $L(\overrightarrow{\boldsymbol{x}}, \overrightarrow{\boldsymbol{x}})$ to implement a simple Newton-Raphson scheme.

5 Inverting $(I - L)$

To implement the acceleration step in equation (7) we need to compute the term $(I - L)^{-1} \cdot \overrightarrow{\boldsymbol{u}}$.

A natural approach would be to convert this term into a (sparse) matrix and use a linear (sparse) algebra solver to get the solution.[7]

In general the complexity of matrix inversion is greater than N^2 and since the jacobian we consider doesn't exhibit a simple structure, matrix inversion quickly becomes very expensive computationally.

[7] For the model FI described below, solving equation (14) with a sparse jacobian takes 40s vs 253ms for our fastest method.

5.1 (Truncated) Neumann Series

A fruitful idea consists in considering the Neumann identity:

$$(I - L)^{-1} \cdot \overrightarrow{\boldsymbol{u}} = (I + L + L^2 + \ldots) \cdot \overrightarrow{\boldsymbol{u}}. \tag{14}$$

Recall that close to the solution, spectral radius of $\mathcal{T}'$ hence of L is strictly smaller than one for any well defined model. For this reason, the infinite sum in equation (14) is converging and well defined.

For any integer p let us define the partial sum $\overrightarrow{\boldsymbol{v}}_p = \sum_k^{p-1} L^k \cdot \overrightarrow{\boldsymbol{u}}$. Given a vector norm $|\cdot|$, the approximation error then takes a very simple form:

$$\begin{aligned} \big|(I - L) \cdot \overrightarrow{\boldsymbol{v}}_p - \overrightarrow{\boldsymbol{u}}\big| &= \Big|\sum_{k=0}^{p-1} L^k \cdot \overrightarrow{\boldsymbol{u}} - \sum_{k=1}^{p} L^k \cdot \overrightarrow{\boldsymbol{u}} - \overrightarrow{\boldsymbol{u}}\Big| \\ &= \big|L^p \cdot \overrightarrow{\boldsymbol{u}}\big| \\ &\leq C\left(\rho(L)\right)^p |\overrightarrow{\boldsymbol{u}}| \end{aligned} \tag{15}$$

for some constant $C > 0$.[8] Since we don't know the constant, we thus can't predict in advance how many terms are required to achieve a given precision.

In practice, the Neumann sum can be computed recursively by setting $\overrightarrow{\boldsymbol{v}}_0 = \overrightarrow{\boldsymbol{u}}$ and by computing recursively the terms

$$\overrightarrow{\boldsymbol{v}}_{p+1} = \overrightarrow{\boldsymbol{u}}_0 + L \cdot \overrightarrow{\boldsymbol{v}}_p \tag{16}$$

until $|\overrightarrow{\boldsymbol{v}}_p - \overrightarrow{\boldsymbol{v}}_{p-1}| < \eta$ for a given $\eta > 0$. Since we have $\overrightarrow{\boldsymbol{v}}_p - \overrightarrow{\boldsymbol{v}}_{p-1} = L^p \overrightarrow{\boldsymbol{u}}$, the procedure stops when equation (15) is solved up to precision η.

In our applications, it turns out that computing the jacobian very precisely is not needed in the first iterations of the outer algorithm. Then to speed up the calculations, we can limit the number of terms to a predetermined number Q. This often leads to faster model solution than solving the system completely. We refer to this variant as the *optimistic* Neumann sum.

5.2 Krylov Projection / GMRES

Instead of the optimistic truncating heuristics, we can also consider the sum

$$\overrightarrow{\boldsymbol{w}}_Q = c_0 \overrightarrow{\boldsymbol{u}} + c_1 L \cdot \overrightarrow{\boldsymbol{u}} + \ldots + c_{Q-1} L^{Q-1} \cdot \overrightarrow{\boldsymbol{u}}$$

defined for the scalar coefficients $c = (c_0, \ldots, c_{Q-1})$ and look for the coefficients that minimize

$$\big|(I - L)\overrightarrow{\boldsymbol{w}}_Q - \overrightarrow{\boldsymbol{u}}\big|. \tag{17}$$

When $|\cdot|$ is a Hilbert norm associated with a scalar product defined over $\mathcal{X}$, the result of the minimization problem is precisely the orthogonal projection of $\overrightarrow{\boldsymbol{u}}$ over the Krylov subspace spanned by Q iterates of L:

$$\mathcal{K}(\overrightarrow{\boldsymbol{u}}, L, Q) = \operatorname{span}\big(\overrightarrow{\boldsymbol{u}}, L\overrightarrow{\boldsymbol{u}}, \ldots, L^{Q-1}\overrightarrow{\boldsymbol{u}}\big). \tag{18}$$

By construction, the resulting projection is a better minimizer of equation (17) than the truncated sum $\overrightarrow{\boldsymbol{v}}_Q$. In practice, it often converges in fewer iterations than the Neumann sum. But in theory, the classical approximation bound for general nonsymmetric operators is essentially[9] the same as for the truncated Neumann sum, that is $C\left(\rho(L)\right)^Q |\overrightarrow{\boldsymbol{u}}|$.

The Krylov projection method we just described can be implemented using the famous GMRES algorithm that is designed to solve in $\overrightarrow{\boldsymbol{v}}$ the linear system $(I - L)\overrightarrow{\boldsymbol{v}} = \overrightarrow{\boldsymbol{u}}$. It works by solving the same minimization problem as in equation (17), on the Krylov subspace $\mathcal{K}((I - L)\overrightarrow{\boldsymbol{u}}_0 - \overrightarrow{\boldsymbol{u}}_0, I - L, Q)$ for any given $\overrightarrow{\boldsymbol{u}}_0$. The

[8] Using the supremum norm on linear operators $\|M\| = \sup_{|x|=1} |Mx|$, we can readily obtain the bound $|(I-L)\cdot\overrightarrow{\boldsymbol{v}}_Q - \overrightarrow{\boldsymbol{u}}| \leq \|L\|^k \overrightarrow{\boldsymbol{u}}$ but that one is of limited use since the operator norm and the spectral radius are not directly related in general. Instead, Gelfand's formula $\|M^k\|^{1/k} \to_{k\to\infty} \rho(A)$ yields the required result.

[9] One difference is that when Q is equal to the number of dimensions of $\mathcal{X}^N$, the Krylov basis spans the whole space, and the error is then zero after a finite (but potentially long) number of iterations.

minimization is done recursively for all $Q' \leq Q$ and several optimization techniques are used to make it numerically stable and memory efficient (see Saad and Schultz (1986) for details).

For the choice $\overrightarrow{\boldsymbol{u}}_0 = \overrightarrow{\boldsymbol{u}}$ this Krylov space exactly corresponds to $\mathcal{K}(\overrightarrow{\boldsymbol{u}}, L, Q)$ meaning that the GMRES algorithm will produce the same result as the minimization problem from equation (17) for any given Q.

Several libraries implement a GMRES function that we can use to obtain a more precise estimate of the inverse than the Neumann sum. It comes at the cost of higher computational complexity and memory footprint, since all base vectors must be computed and stored. This can limit the value of Q that can be used.[10]

6 Newton-Krylov Method

There is a ready alternative to the (accelerated) time iteration algorithm exposed in Section 3. It consists in solving the equation $G(\overrightarrow{\boldsymbol{x}}) = 0$ using a nonlinear solution algorithm. We describe below the Newton-Raphson algorithm, which is the most natural choice for this problem.

Starting with an initial guess, $\overrightarrow{\boldsymbol{x}}_k$, we can compute the residuals $\overrightarrow{\boldsymbol{r}}_k = G(\overrightarrow{\boldsymbol{x}}_k)$. The Newton formula provides the new estimate:

$$\overrightarrow{\boldsymbol{x}}_{k+1} = \overrightarrow{\boldsymbol{x}}_k - \underbrace{G'\big(\overrightarrow{\boldsymbol{x}}_k\big)^{-1} \cdot \overrightarrow{\boldsymbol{r}}_k}_{\overrightarrow{\boldsymbol{\delta}}_k}$$

where $\overrightarrow{\boldsymbol{\delta}}_k$ is the Newton step. As is well known, this algorithm also converges *quadratically*.

The main computational burden of the Newton-Raphson algorithm is the inversion of the jacobian $G'(\overrightarrow{\boldsymbol{x}}_k)$ required to solve in $\overrightarrow{\boldsymbol{\delta}}_k$ the system

$$G'\big(\overrightarrow{\boldsymbol{x}}_k\big) \cdot \overrightarrow{\boldsymbol{\delta}}_k = \overrightarrow{\boldsymbol{r}}_k. \tag{19}$$

As noted above, it is too complicated for direct linear algebra methods on full or sparse matrices and we need to implement some iterative method instead. From equation (4) we know

$$G'\big(\overrightarrow{\boldsymbol{x}}_k\big) = F'_A\big(\overrightarrow{\boldsymbol{x}}_k, \overrightarrow{\boldsymbol{x}}_k\big) + F'_B\big(\overrightarrow{\boldsymbol{x}}_k, \overrightarrow{\boldsymbol{x}}_k\big) \tag{20}$$

meaning that we can efficiently evaluate $G'(\overrightarrow{\boldsymbol{x}}_k) \cdot \overrightarrow{\boldsymbol{u}}$ for any vector $\overrightarrow{\boldsymbol{u}}$ using the same steps from Section 4. We can thus implement the GMRES algorithm to solve the minimization problem

$$\min_{\overrightarrow{\boldsymbol{u}} \in \mathcal{K}(\overrightarrow{\boldsymbol{u}}_0, G'(\overrightarrow{\boldsymbol{x}}_k), Q)} \big\| G'\big(\overrightarrow{\boldsymbol{x}}_k\big) \cdot \overrightarrow{\boldsymbol{u}} - \overrightarrow{\boldsymbol{r}}_k \big\| \tag{21}$$

leading to a solution algorithm known as a Newton-Krylov method (or nonlinear GMRES).

In general we don't have any information about the spectral radius of operator $G'(\overrightarrow{\boldsymbol{x}}_k)$ and can't derive useful error bounds for equation (21). As a result, the dimension Q required to meet a prespecified error level might be arbitrarily high.

6.1 Preconditioning

Equation (21) can be preconditioned, leading to more efficient inversion of the linear system. Let us premultiply both sides of equation (19) by $F'_A(\overrightarrow{\boldsymbol{x}}_k, \overrightarrow{\boldsymbol{x}}_k)^{-1}$ to get the equivalent system:

$$\big(F'_A(\overrightarrow{\boldsymbol{x}}_k, \overrightarrow{\boldsymbol{x}}_k)^{-1}(F'_A + F'_B)\big) \cdot \overrightarrow{\boldsymbol{r}}_k = F'_A(\overrightarrow{\boldsymbol{x}}_k, \overrightarrow{\boldsymbol{x}}_k)^{-1}\overrightarrow{\boldsymbol{r}}_k$$

and rewrite it as

$$\big(I + F'_A(\overrightarrow{\boldsymbol{x}}_k, \overrightarrow{\boldsymbol{x}}_k)^{-1}F'_B\big) \cdot \overrightarrow{\boldsymbol{r}}_k = F'_A(\overrightarrow{\boldsymbol{x}}_k, \overrightarrow{\boldsymbol{x}}_k)^{-1}\overrightarrow{\boldsymbol{r}}_k$$

or using the notations from Section 3.2

$$\big(I - L(\overrightarrow{\boldsymbol{x}}_k, \overrightarrow{\boldsymbol{x}}_k)\big) \cdot \overrightarrow{\boldsymbol{\delta}}_k = F'_A(\overrightarrow{\boldsymbol{x}}_k, \overrightarrow{\boldsymbol{x}}_k)^{-1}\overrightarrow{\boldsymbol{r}}_k. \tag{22}$$

Since we know that the spectral radius of $L(\overrightarrow{\boldsymbol{x}}_k, \overrightarrow{\boldsymbol{x}}_k)$ is smaller than one close to the solution, we can use the Neumann series or Krylov projection methods from Section 5.2 to invert $I - L$. In particular, we get similar theoretical bounds on the convergence rate in $C\,\rho(L)^k$ for some $C > 0$.

[10] Our implementation uses basis orthonormalization and Givens rotation to speed up the least-square step from the minimization problem. In our applications, we haven't found any computational advantage in *restarting* iterations (replacing $\overrightarrow{\boldsymbol{u}}_0$ by the solution at order Q) as long as memory is sufficient to contain the full base.

6.2 Comparison with Accelerated Time Iterations

When the initial guess is close enough to the solution, the Newton-Krylov algorithm converges at a *quadratic* rate, like the accelerated time iteration algorithm. It does not produce the same iterates as the latter however. This can be seen directly by noting that the former evaluates derivatives of F at $(\overrightarrow{\boldsymbol{x}}_k, \overrightarrow{\boldsymbol{x}}_k)$ while the latter evaluates them at $(\overrightarrow{\boldsymbol{x}}_{k+1}, \overrightarrow{\boldsymbol{x}}_k)$. From a mathematical perspective, Section B.3 of the appendix shows that accelerated time iterations are mathematically equivalent to Newton-Raphson applied to the function $\overrightarrow{\boldsymbol{u}} \to \overrightarrow{\boldsymbol{u}} - T(\overrightarrow{\boldsymbol{u}})$ instead of $u \to F(\overrightarrow{\boldsymbol{u}}, \overrightarrow{\boldsymbol{u}})$.

Since these two functions are distinct, we can't predict generically which of the two generated sequences will solve models faster if at all: actual differences in convergence speed will be driven by the first iterates for which the asymptotic estimates are not informative. Hence we expect relative performance and usability of both methods to be model dependent. Still, for a given number of iterations on k, accelerated time-iterations have a higher computational complexity, since a nonlinear system must be solved to compute $\mathcal{T}(\overrightarrow{\boldsymbol{x}}_k)$.

6.3 Safeguarded Newton Steps

In order to improve global convergence properties, the first iterates of Newton-Krylov algorithm are typically refined to ensure that the optimization progresses. We have implemented a classical backstepping procedure to that effect. The method requires two parameters: contraction factor $\alpha \in]0, 1[$ (that we set to 0.5) and the Armijo constant $c_1 > 0$ (that we set to 10^{-4}).

After the step $\overrightarrow{\boldsymbol{\delta}}_k$ has been computed, we consider the family of new potential iterates defined by $\overrightarrow{\boldsymbol{x}}_k + \lambda \overrightarrow{\boldsymbol{\delta}}_k$ for $\lambda \in [0, 1]$ and define $\xi(\lambda) = |F(\overrightarrow{\boldsymbol{x}}_k + \lambda \overrightarrow{\boldsymbol{\delta}}_k, \overrightarrow{\boldsymbol{x}}_k + \lambda \overrightarrow{\boldsymbol{\delta}}_k)|$. We say that iterate defined by a given λ is making progress if it satisfies the Armijo condition $\xi(\lambda) < \xi(0)\sqrt{1 - 2c_1\alpha}$ (see Kelley (1995) for details). The new iterate is then obtained by choosing $\lambda_i = \alpha^i$ for the smallest $i \geq 0$ such that the Armijo condition is satisfied.

Asymptotically, when the current guess is close enough to the solution, the full Newton step ($\lambda = 1$) will satisfy the Armijo condition. Hence the safeguard does not alter the quadratic convergence rate of the algorithm. But in the first iterations, it avoids overshooting and divergence, removing the need to have a very precise initial guess.

7 Timings

We benchmark the methods that we have described on three simple models that can be solved using the time iteration method. Section A provides the precise description of all models, as well as the procedure to discretize them in order to comply with the specification from Section 2.1. Their main features as well as some parameters of the algorithm are reported in Table 1.

The first model is an infinite horizon consumption-savings (CS) model with a borrowing constraint. The specification is the same as in L. Maliar, S. Maliar, and Winant (2021). It features an exogenous persistent income process, discretized as a 3 states Markov chain, a single continuous state (cash in hand) discretized with 200 points and a single control (consumption). In order to deal with the occasionally binding constraint we reformulate the Euler equation using the Fisher Burmeister transform and add an auxiliary control for the Lagrange multiplier (see L. Maliar, S. Maliar, and Winant (2021)). Future variables are interpolated outside of the grid using linear splines.

The second model is a standard Real Business Cycle model (RBC). It features two continuous states (capital and productivity). The state-space is discretized as a 50×50 Cartesian grid. The two controls (consumption and labour) are interpolated outside of the grid using cubic splines. The i.i.d. exogenous shock to productivity is discretized using 5 Gauss-Hermite nodes.

The third model is a two countries neoclassical economy taken from Coeurdacier, Rey, and Winant (2020), with Epstein-Zin preferences, capital adjustment costs and a riskless bond (FI). It features three continuous states (capital in both countries and net foreign asset position) discretized as a $50 \times 50 \times 50$ Cartesian grid and two exogenous states (productivity in both countries), each of them discretized as a 3 states Markov chain. The model is reformulated as a set of first order equations by using auxiliary controls for the Epstein-Zin

	CS	RBC	FI
N. of continuous states	1	2	3
N. of grid points	3×200	50^2	9×50^3
Initial Iterations	0	25	5
Interpolation Type	Linear	Cubic	Cubic
N. of CPUs	1	4	8

Table 1: Summary of the three models

	CS	RBC	FI
Initial Refinement	-	277ms	187s
Time Iteration	102ms	820ms	6720 s
Accelerated Time Iteration			
Full	55ms	361ms	1762 s
Optimistic	25ms	117ms	623 s
GMRES	26ms	143ms	491 s
Newton Krylov			
Full	72ms	411ms	1506 s
Optimistic	27ms	91ms	480 s
GMRES	33ms	127ms	380 s

Table 2: Timing Comparison: Initial Refinement + Model Solution

values and expected values. We approximate the 8 controls of the model using cubic splines.[11]

Each model has a very simple initial guess for the controls. When that one is too far from the solution, the accelerated time iterations and/or the Newton-Krylov algorithm can fail. For this reason, we perform a fixed initial number of time-iteration steps, and use the result as a refined guess to benchmark all algorithms against each other. The number of initial iterations is reported with other model statistics in Table 1.

All algorithms are coded using the Julia language and run on an Intel Core i7-9700 processor. They all use the same elementary functions so that we can reliably comment on algorithmic gains.[12] The implementation extensively uses Julia's ability to differentiate automatically any piece of code and just-in-time compile the resulting graph. Accordingly, we exclude all compilation tasks from the timings by performing a *warmup* call of the solution functions, before running the benchmarks.

Table 2 reports the running time for the unmodified time iteration algorithm (Section 3.1), for accelerated time iteration (Section 3.2) and Newton-Krylov (Section 6).[13] For the latter two, we compare three methods to invert the jacobian: by computing the Neumann sum until convergence (*Full*), by truncating this sum with $Q = 50$ terms (*Optimistic*), or by using the GMRES algorithm (*GMRES*) also with at most $Q = 50$ base vectors.

With the Neumann jacobian inversion method, both Accelerated Time Iterations and Newton-Krylov provide significant performance gains over the reference Time Iteration Algorithm. We measure speedups around $\times 1.5$, $\times 4$ and $\times 5$ for the CS, RBC and FI models respectively. Similar timings between the two methods echo the fact that Accelerated Time Iteration is mathematically equivalent to a Newton scheme applied to find the fixed point of the time iteration operator.

Table 3 shows a breakdown of the main function calls, for the FI model. Performance gains stem from a drastic reduction in the number of model evaluations. In the *Full* accelerated algorithm, F (resp. F') is evaluated 33 times (resp. 16) instead of 1585 times (resp. 463 times) for the non-accelerated version. Instead, these are replaced by calls to the cheaper linear operator L to approximate the actual time-iterations.

[11] This seemingly reasonable model features 1 125 000 states and the 9 000 000 elements of the control vector, stored as double precision floats, take about 72MB of memory. With this size of model, naive approaches can incur a significant computational cost.

[12] Since the language allows us to define arrays of arbitrary types and efficient iterators, the memory layout and the model's computational representation closely mimic the objects described in the algorithmic section. Code is available on github and from the author's website.

[13] All algorithms have the same termination criterion: they terminate when equation (4) is solved up to accuracy $\epsilon = 10^{-8}$ for the supremum norm.

	F	F'_A	F'_B	$L \cdot u$
Time Iteration	1585	458	–	–
A.T.I. (*full*)	33	16	5	1710
A.T.I. (*optimistic*)	45	22	8	400
A.T.I. (*gmres*)	33	15	8	250
Newton (*full*)	6	5	5	1570
Newton (*optimistic*)	9	8	8	400
Newton (*gmres*)	6	5	5	250

Table 3: Number of Function Calls for Model FI

	CS	RBC	FI
Time Iteration	102 ms	1097 ms	6907s
Refinement + Newton-Krylov			
GMRES (Q=50)	33ms	404ms	567 s
Safeguarded Newton Krylov			
GMRES (Q=50, no preconditioning)	31ms	1940ms	-
GMRES (Q=50)	33ms	446ms	455 s
GMRES (Q=25)	18ms	242ms	341 s

Table 4: Timing Comparison: Direct Solution

Limiting the number of evaluations of L at the cost of a more imprecise jacobian inversion, also clearly improves running times with $\times 2$ or $\times 3$ speedups for the *Optimistic* variant. For the same number of evaluations of L, the *GMRES* variant produces more accurate jacobian, limiting the number of outer iterations and the total running time. Interestingly the cost of storing the Krylov basis and computing all dot products does not seem to be a limiting factor even for the bigger model of the three.

Instead of focusing on asymptotic speed, we can also consider the direct solution of the model, without the initial refinement phase. Table 4 reports the timings for the Newton-Krylov method with safeguarded steps with different parameters of the GMRES inversion. First we observe that the preconditioning step is important: for the RBC model, trying to solve the model directly without preconditioning leads to very poor performance and even fails for the FI model.[14] Otherwise the simple backstepping procedure, produces robust convergence and leads to the fastest solution method overall if we compare it to the total time of the alternatives (that is if we add to table 3 the time of the initial refinement step). It also allows us to reduce the size of the Krylov basis without harming convergence and speeding up the computations even further.

8 Conclusion

The life of a computational economist is plagued by a very annoying problem: a realistic nonlinear rational expectations model can take a long time to solve, with the commonly used time iteration algorithm.

In this paper we have proposed a method to accelerate convergence with a scheme that is reminiscent of the Howard improvement method, whereby policy improvement steps (here the time iterations) alternate with value evaluation steps (here the gradient of the time iterations).

We have shown how to implement it efficiently by computing explicitly all derivative operators and discussed several ways to invert the needed jacobians.

Practical results are encouraging: quadratic convergence leads to very sizable gains. In the biggest model we consider, the fastest method offers a $\times 10$ speedup over the benchmark algorithm in terms of asymptotic speed. In fact, in line with the theory, our algorithm performs similarly to a global Newton-Raphson algorithm applied to the model equations.

[14]Detailed profiling data is available in appendix C. Note that reported memory allocations correspond to variables that are created on the heap (as with C's 'malloc' function) and subsequently freed by Julia's garbage collector. The time allotated to the garbage collection is part of the total reported time and is indeed very small. In fact most of our computations avoid the use of such variables: they are done inplace in a preallocated workplace whose creation also takes a negligible time.

We also demonstrate that similar computation steps can be used to implement a safeguarded Newton-Krylov method that can solve models directly from a naive initial guess. It turns out to be the fastest solution method overall for our most challenging model, leading to a $\times 20$ speedup.

One should be careful about not overgeneralizing the timings we present. While we have been careful to make a fair comparison between the various algorithms, we can't guarantee that implementations are the fastest possible. In fact, there are almost surely performance gains that can be obtained by improving memory allocation and code vectorization. A nice feature of our family of algorithms is that they spend most of their time inverting an abstract linear operator that has a simple memory representation. To provide even higher speed-ups, one can thus focus all efforts on improving the efficiency of this particular operation, for instance, by using GPU acceleration.

Finally, the computational representation of the model derivatives can be used for other purposes. For instance, it can be combined with more sophisticated iterative solvers beyond those presented here. In ongoing work, we also use it to study the sensitivity of the solution to the model parameters, as well as to study the spectral properties of the time iteration algorithm.

References

Azinovic, Marlon, Luca Gaegauf, and Simon Scheidegger (2022). "Deep Equilibrium Nets". In: *International Economic Review* 63.4, pp. 1471–1525. DOI: 10.1111/iere.12575. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/iere.12575 (visited on 02/25/2025).

Brumm, Johannes and Simon Scheidegger (May 22, 2017). *Using Adaptive Sparse Grids to Solve High-Dimensional Dynamic Models*. DOI: 10.2139/ssrn.2349281. URL: https://papers.ssrn.com/abstract=2349281 (visited on 02/25/2025). Pre-published.

Carroll, Christopher D. (June 1, 2006). "The Method of Endogenous Gridpoints for Solving Dynamic Stochastic Optimization Problems". In: *Economics Letters* 91.3, pp. 312–320. DOI: 10.1016/j.econlet.2005.09.013. URL: https://www.sciencedirect.com/science/article/pii/S0165176505003368 (visited on 02/25/2025).

Coeurdacier, Nicolas, Hélène Rey, and Pablo Winant (June 1, 2020). "Financial Integration and Growth in a Risky World". In: *Journal of Monetary Economics* 112, pp. 1–21. DOI: 10.1016/j.jmoneco.2019.01.022. URL: https://www.sciencedirect.com/science/article/pii/S0304393218300333 (visited on 02/25/2025).

Coleman, Wilbur John (1990). "Solving the Stochastic Growth Model by Policy-Function Iteration". In: *Journal of Business & Economic Statistics* 8.1, pp. 27–29. DOI: 10.2307/1391745. JSTOR: 1391745. URL: https://www.jstor.org/stable/1391745 (visited on 02/25/2025).

Deaton, Angus and Guy Laroque (1992). "On the Behaviour of Commodity Prices". In: *The Review of Economic Studies* 59.1, pp. 1–23. DOI: 10.2307/2297923. eprint: https://academic.oup.com/restud/article-pdf/59/1/1/4356717/59-1-1.pdf. URL: https://doi.org/10.2307/2297923.

Den Haan, Wouter J. and Albert Marcet (1990). "Solving the Stochastic Growth Model by Parameterizing Expectations". In: *Journal of Business & Economic Statistics* 8.1, pp. 31–34. DOI: 10.2307/1391746. JSTOR: 1391746. URL: https://www.jstor.org/stable/1391746 (visited on 02/25/2025).

Fernández-Villaverde, Jesús et al. (Aug. 1, 2015). "Nonlinear Adventures at the Zero Lower Bound". In: *Journal of Economic Dynamics and Control* 57, pp. 182–204. DOI: 10.1016/j.jedc.2015.05.014. URL: https://www.sciencedirect.com/science/article/pii/S0165188915000949 (visited on 02/25/2025).

Judd, Kenneth L. (1998). *Numerical Methods in Economics*. Vol. 1. MIT Press Books. The MIT Press. URL: https://ideas.repec.org/b/mtp/titles/0262100711.html.

Judd, Kenneth L., Lilia Maliar, Serguei Maliar, and Inna Tsener (2017). "How to Solve Dynamic Stochastic Models Computing Expectations Just Once". In: *Quantitative Economics* 8.3, pp. 851–893. DOI: 10.3982/QE329. URL: https://onlinelibrary.wiley.com/doi/abs/10.3982/QE329 (visited on 02/25/2025).

Judd, Kenneth L., Lilia Maliar, Serguei Maliar, and Rafael Valero (July 1, 2014). "Smolyak Method for Solving Dynamic Economic Models: Lagrange Interpolation, Anisotropic Grid and Adaptive Domain". In: *Journal of Economic Dynamics and Control* 44, pp. 92–123. DOI: 10.1016/j.jedc.2014.03.003. URL: https://www.sciencedirect.com/science/article/pii/S0165188914000621 (visited on 02/25/2025).

Kelley, C. T. (1995). "Global Convergence". In: *Iterative Methods for Linear and Nonlinear Equations*. Frontiers in Applied Mathematics 16. Philadelphia, PA: Society for Industrial and Applied Mathematics. Chap. 8. DOI: 10.1137/1.9781611970944.

Maliar, Lilia, Serguei Maliar, and Pablo Winant (Sept. 1, 2021). "Deep Learning for Solving Dynamic Economic Models." In: *Journal of Monetary Economics* 122, pp. 76–101. DOI: 10.1016/j.jmoneco.2021.07.004. URL: https://www.sciencedirect.com/science/article/pii/S0304393221000799 (visited on 08/08/2024).

Maliar, Serguei, Lilia Maliar, and Kenneth Judd (Feb. 1, 2011). "Solving the Multi-Country Real Business Cycle Model Using Ergodic Set Methods". In: *Journal of Economic Dynamics and Control*. Computational Suite of Models with Heterogeneous Agents II: Multi-Country Real Business Cycle Models 35.2, pp. 207–228. DOI: 10.1016/j.jedc.2010.09.014. URL: https://www.sciencedirect.com/science/article/pii/S0165188910002186 (visited on 02/25/2025).

Puterman, Martin L. and Shelby L. Brumelle (1979). "On the Convergence of Policy Iteration in Stationary Dynamic Programming". In: *Mathematics of Operations Research* 4.1, pp. 60–69. JSTOR: 3689239. URL: https://www.jstor.org/stable/3689239 (visited on 02/25/2025).

Rouwenhorst, K. Geert (1995). "Asset Pricing Implications of Equilibrium Business Cycle Models". In: *Frontiers of Business Cycle Research*. Ed. by Thomas F. Cooley. Princeton University Press, pp. 294–330. DOI: 10.2307/j.ctv14163jx.16. JSTOR: j.ctv14163jx.16. URL: http://www.jstor.org/stable/j.ctv14163jx.16 (visited on 03/15/2025).

Rust, John (Jan. 1, 1996). "Chapter 14 Numerical Dynamic Programming in Economics". In: *Handbook of Computational Economics*. Ed. by Thomas F. Cooley. Vol. 1. Elsevier, pp. 619–729. DOI: 10.1016/S1574-0021(96)01016-7. URL: https://www.sciencedirect.com/science/article/pii/S1574002196010167 (visited on 02/25/2025).

Saad, Youcef and Martin H. Schultz (1986). "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems". In: *SIAM Journal on Scientific and Statistical Computing* 7.3, pp. 856–869. DOI: 10.1137/0907058. eprint: https://doi.org/10.1137/0907058. URL: https://doi.org/10.1137/0907058.

A Models

This appendix provides a detailed description of the three models used in the main text. This should be enough to replicate the results without any ambiguity. For economic justifications we refer the readers to the source papers.

A.1 Consumption Savings (CS)

The specification of the consumption-saving model is the same as in L. Maliar, S. Maliar, and Winant (2021). Calibration is provided in Table 5.

The vector of states is $s_t = (y_t, w_t)$ where y_t is log-income and w_t is available income.

The vector of controls to be determined in any state is $x_t = (c_t, h_t)$ where c_t denotes consumption and h_t is a budget multiplier. It is strictly positive if $c_t = w_t$ and zero when $c_t < w_t$.

Parameter	Value
Time-discount	$\beta = 0.96$
Risk-aversion	$\gamma = 4.0$
Risk-free interest rate	$r = 1.02$
Income Process (autocorr.)	$\rho = 0.9$
Income Process (stdev.)	$\sigma = 0.1$

Table 5: Calibration (CS)

A.1.1 State transitions

Income y_t follows an exogenous AR1 process with autocorrelation ρ and standard deviation σ.

Available income evolves according to the rule:

$$w_t = \exp(y_t) + (w_{t-1} - c_{t-1})r.$$

A.1.2 Optimality conditions

Optimal consumption is determined by the Euler condition

$$\beta \mathbb{E}_t\Big[\Big(\frac{c_{t+1}}{c_t}\Big)^{-\gamma} r\Big] - 1 + h_t = 0$$

with multiplier h_t satisfying the condition

$$\mathrm{FB}(h_t, w_t - c_t) = 0$$

where $\mathrm{FB}(a, b) = a + b - \sqrt{a^2 + b^2}$ is the Fisher-Burmeister function.

A.1.3 Approximation

The process (y_t) is discretized as a 3 states Markov chain, using Rouwenhorst's algorithm (Rouwenhorst, 1995). This yields an exogenous state-space $\mathcal{S}^{\text{exo}} = (\overline{\boldsymbol{s}}^n)_{n \in \{1 \dots N^{\text{exo}}\}}$ with $N^{\text{exo}} = 3$ discrete exogenous values and the corresponding transition matrix $P \in \mathbb{R}^{N^{\text{exo}} \times N^{\text{exo}}}$.

The endogenous state-space is $\mathcal{S}^{\text{endo}} = [0.5, 4]$. It is discretized with $N^{\text{endo}} = 200$ linearly spaced points.

Finally the discretized state-space of the model is defined as the Cartesian product of both exogenous and endogenous state spaces $\overline{\boldsymbol{s}} = \overline{\boldsymbol{s}}^{\text{exo}} \times \overline{\boldsymbol{s}}^{\text{endo}} = (\overline{\boldsymbol{s}}^p, \overline{\boldsymbol{s}}^q)_{p \in \{1 \dots N^{\text{exo}}\}, q \in \{1 \dots N^{\text{endo}}\}}$. With linear indexing we can also write it as $(\overline{\boldsymbol{s}}^n)_{n \in \{1 \dots N\}}$ where $N = N^{\text{exo}} N^{\text{endo}} = 600$ is the number of grid points.

The initial guess $\overrightarrow{\boldsymbol{x}}_0 = (\overrightarrow{\boldsymbol{x}}^n[0])_{n \in \{1 \dots N\}} = ((c_n, h_n))_{n \in \{1 \dots N\}}$ on the grid satisfies the relation

$$\begin{aligned} c_n &= \min(0.95 w_n, 1.0 + 0.04(w_n - 1.0)), \\ h_n &= 0.05. \end{aligned}$$

Future controls between grid points are interpolated using linear interpolation.

A.2 Real Business Cycle (RBC)

The model is taken straight from Dynare's example files.

The vector of states is $s_t = (k_t, a_t)$ where k_t is the capital level and a_t is the productivity level.

We treat all other variables as controls even though some of them could be more efficiently computed in closed form.

The vector of controls is thus

- i_t: investment
- n_t: Labor level
- w_t: wage
- c_t: consumption
- $r_{k,t}$: return on capital

The only exogenous shock is productivity innovation z_t. It is i.i.d. and follows a normal law with standard deviation σ_z.

The parameters and the steady-state values of the model are given in Tables 6 and 7.

Parameter	Value
Time Discount	$\beta = 0.99$
Capital Depreciation	$\delta = 0.025$
Capital Share	$\alpha = 0.33$
E.I.S	$1/\sigma = 1/2.0$
Labor Preference Weight	$\chi = w/c^\sigma/n^\eta$
Frisch Elasticity	$\frac{1}{\eta} = 1.0$
Productivity (autocorr.)	$\rho = 0.8$
Productivity (stdev.)	$\sigma_z = 0.001$

Table 6: Calibration (RBC)

Variable	Value
Labor	$\overline{n} = 0.33$
Productivity	$\overline{z} = 1.0$
Capital	$\overline{k} = n/(r_k/\alpha)^{1/(1-\alpha)}$
Investment	$\overline{i} = \delta k$
Consumption	$\overline{c} = zk^\alpha n^{1-\alpha} - i$

Table 7: Steady-state Values (RBC)

A.2.1 Equations

The transitions of the states are given by:

$$k_t = (1-\delta)k_{t-1} + i_{t-1}$$
$$a_t = \rho a_{t-1} + z_t$$

Optimality conditions are:

$$\begin{aligned}
1 &= \mathbb{E}_t\Big[\beta\left(\frac{c_t}{c_{t+1}}\right)^{\sigma}(1-\delta+r_{k,t+1})\Big] \\
w_t &= \chi n_t^{\eta} c_t^{\sigma} \\
c_t &= \exp(a_t)k_t^{\alpha}n_t^{1-\alpha} - i_t \\
r_{k,t} &= \alpha\exp(a_t)\left(\frac{n_t}{k_t}\right)^{1-\alpha} \\
w_t &= (1-\alpha)\exp(a_t)\left(\frac{k_t}{n_t}\right)^{\alpha}
\end{aligned}$$

A.2.2 Discretization

The state-space is defined as $\mathcal{S} = [0.5\overline{k}, 2\overline{k}] \times [-0.1, 0.1]$. It is then discretized as a Cartesian grid $\overline{\boldsymbol{s}} = (\overline{\boldsymbol{s}}^n)_{n\in\{1\dots N\}}$ using 50 points in each dimension. We have $N = 2500$ points.

The initial guess on the grid is equal to steady-state values $\overrightarrow{\boldsymbol{x}} = (\overrightarrow{\boldsymbol{x}}^n)_{n=1:N} = ((\overline{i}, \overline{n}, \overline{w}, \overline{c}, \overline{r}))_{n\in\{1\dots N\}}$.

We discretize the exogenous shock z using $K = 5$ Gauss-Hermite nodes $(z_k)_{k=1:K}$ with associated weights $(w_k)_{k\in\{1\dots K\}}$. For a given state $\overline{\boldsymbol{s}}^n$ and a corresponding control, the transition function τ is:

$$\tau(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n) = (w_k, g(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}^n, z_k))_{k\in\{1\dots K\}}.$$

Future controls are approximated using a cubic spline approximation.

A.3 Financial Integration (FI)

The model is taken from Coeurdacier, Rey, and Winant (2020).

A.3.1 Definitions

The vector of states contain 5 variables $s_t = (A_{1,t}, A_{2,t}, k_{1,t}, k_{2,t}, b_{f,t})$.

- vector of exogenous states $s_t^{\text{exo}} = (A_{1,t}, A_{2,t})$
  - $A_{1,t}, A_{2,t}$: productivity level in both countries
- vector of endogenous states $s_t^{\text{endo}} = (k_{1,t}, k_{2,t}, b_{f,t})$
  - $k_{1,t}, k_{2,t}$: capital levels in each country
  - $b_{f,t}$: net position of country 1

The vector of controls contains 8 variables $x_t = (b'_{f,t}, p_{f,t}, i_{1,t}, i_{2,t}, \tilde{w}_{1,t}, \tilde{w}_{2,t}, w_{1,t}, w_{2,t})$ where:

- $b'_{f,t}$ is the net position chosen in period $t-1$ for period t
- $p_{f,t}$ is the price of a two period bond yielding in the next period
- $i_{1,t}$ and $i_{2,t}$ is investment in each country
- $w_{1,t}$ and $w_{2,t}$ is the value function of each country
- $\tilde{w}_{1,t}$ and $\tilde{w}_{2,t}$ are expectation functions appearing in the Epstein-Zin definition of value[15]

Parameters and steady-state values are reported in Tables 8 and 9.

As a function of states and controls at date t, we define several auxiliary variables the same date. They can be directly calculated and substituted in all other equations. We report them in Table 10.

[15]The Epstein-Zin value in country i satisfies the recursive equation $w_{i,t}^{1-\psi} = (1-\beta)(c_{i,t})^{1-\psi} + \beta(\mathbb{E}[w_{i,t+1}^{1-\gamma}])^{\frac{1-\psi}{1-\gamma}}$.

Parameter	Value
Time-discount	$\beta = 0.96$
Capital-share	$\theta = 0.3$
Depreciation Rate	$\delta = 0.08$
Risk Aversion	$\gamma = 4.0$
1/EIS	$\psi = 4.0$
Capital adjustment (1)	$\xi = 0.2$
Capital adjustment (2)	$a_2 = \delta^{\xi}$
Capital adjustment (3)	$a_1 = \delta - a_2/(1-\xi)\delta^{1-\xi}$
Tfp (autocorr)	$\rho_A = 0.9$
Tfp (stdev)	$\sigma_{A,1} = 0.025$

Table 8: Calibration (FI)

Steady-state	Value (country j)
Debt position	$b'_f = b_f = 0$
Price of bond	$p_f = \beta$
Investment	$i_1 = ((1/\beta - (1-\delta))/\theta)^{1/(\theta-1)}\delta$
Value	$w_1 = c_1$
Expected Value	$\tilde{w}_1 = c_1$
Productivity	$A_1 = 0$
Capital	$k_1 = i_1/\delta$
	$b_f = 0$
Production	$y_1 = k_1^{\theta}$
Consumption	$c_1 = (y_1 - \delta k_1)$
Capital Adjustment	$\Phi_1 = a_1 + a_2/(1-\xi)(i_1/k_1)^{1-\xi}$
Capital Adjustment (diff)	$\Phi'_1 = a_2(i_1/k_1)^{-\xi}$

Table 9: Steady-state (FI)

A.3.2 Transition of the states

Exogenous states $A_{1,t}$ and $A_{2,t}$ both follow an AR1 with autocorrelation ρ. They have conditional standard deviation $\sigma_{A,1} = 0.025$ and $\sigma_{A,2} = 0.05$ respectively.

Endogenous states follow the transitions:

$$\begin{aligned} k_{1,t} &= ((1-\delta) + \Phi_{1,t-1})k_{1,t-1} \\ k_{2,t} &= ((1-\delta) + \Phi_{2,t-1})k_{2,t-1} \\ b_{f,t} &= b'_{f,t-1} \end{aligned}$$

This defines a function g such that $s_t^{\text{endo}} = g(s_{t-1}^{\text{exo}}, s_{t-1}^{\text{endo}}, s_t^{\text{exo}})$.

A.3.3 Optimality conditions

The optimality conditions are given by a function f such that $\mathbb{E}_t[f(s_t, x_t, s_{t+1}, x_{t+1})]$ where the components of the residual function f denoted by $(f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8)$ are detailed in the full manuscript equations (omitted here for brevity).

A.3.4 Discretization

The two independent exogenous AR1s $A_{1,t}$ and $A_{2,t}$ are both discretized as 3 states Markov chain, using Rouwenhorst's algorithm. This yields an exogenous state-space $\mathcal{S}^{\text{exo}} = (\overline{\boldsymbol{s}}^{n,\text{exo}})_{n\in\{1\dots N^{\text{exo}}\}}$ with $N^{\text{exo}} = 9$ discrete exogenous values and the corresponding transition matrix $P \in \mathbb{R}^{N^{\text{exo}} \times N^{\text{exo}}}$.

	Country 1	Country 2
Production	$y_{1,t} = k_{1,t}^{\theta} \exp(A_{1,t})$	$y_{2,t} = k_{2,t}^{\theta} \exp(A_{2,t})$
Budget Constraint	$c_{1,t} = y_{1,t} - i_{1,t} + b_{1,t} - p_{f,t} b'_{f,t}$	$c_{2,t} = y_{2,t} - i_{2,t} - (b_{f,t} - p_{f,t} b'_{f,t})$
Investment Friction	$\Phi_{1,t} = a_1 + a_2/(1-\xi)(i_{1,t}/k_{1,t})^{1-\xi}$	$\Phi_{2,t} = a_1 + a_2/(1-\xi)(i_{2,t}/k_{2,t})^{1-\xi}$
Investment Friction (derivative)	$\Phi'_{1,t} = a_2 (i_{1,t}/k_{1,t})^{-\xi}$	$\Phi'_{2,t} = a_2 (i_{2,t}/k_{2,t})^{-\xi}$

Table 10: Auxiliary Variables (FI)

The endogenous state-space is defined as $\mathcal{S}^{\text{endo}} = [0.5\overline{k}_1, 2\overline{k}_1] \times [0.5\overline{k}_2, 2\overline{k}_2] \times [-5, 5]$. It is discretized as a Cartesian grid using 50 points in each dimension. The resulting grid $\overline{\boldsymbol{s}}^{\text{endo}} = (\overline{\boldsymbol{s}}^{n,\text{endo}})_{n \in \{1 \dots N^{\text{endo}}\}}$ thus has $N^{\text{endo}} = 125\,000$ points.

Finally the discretized state-space of the model is defined as the Cartesian product of both exogenous and endogenous state spaces $\overline{\boldsymbol{s}} = \overline{\boldsymbol{s}}^{\text{exo}} \times \overline{\boldsymbol{s}}^{\text{endo}} = (\overrightarrow{\boldsymbol{s}}^{p,\text{exo}}, \overrightarrow{\boldsymbol{s}}^{q,\text{endo}})_{p \in \{1 \dots N^{\text{exo}}\}, q \in \{1 \dots N^{\text{endo}}\}}$. With linear indexing we can also write $(\overline{\boldsymbol{s}}^n)_{n \in \{1 \dots N\}}$. It contains $N = N^{\text{exo}} N^{\text{endo}} = 1\,125\,000$ points.

For a given grid point $\overline{\boldsymbol{s}}^n = (\overrightarrow{\boldsymbol{s}}^{p,\text{exo}}, \overrightarrow{\boldsymbol{s}}^{q,\text{endo}})$ and a vector of controls $\overrightarrow{\boldsymbol{x}}$ the function τ returns N^{exo} values:

$$\tau(\overline{\boldsymbol{s}}^n, \overrightarrow{\boldsymbol{x}}) = (\underbrace{P_{pk}}_{w_k}, g(\overrightarrow{\boldsymbol{s}}^{p,\text{exo}}, \overrightarrow{\boldsymbol{s}}^{q,\text{endo}}, \overrightarrow{\boldsymbol{s}}^{k,\text{exo}}))_{k \in \{1 \dots K\}}.$$

The initial guess on the grid is equal to steady-state values for all grid points $\overline{\boldsymbol{s}}^n$, except for the new financial position, which we set equal to the old one on the grid, denoted here by $b_{f,n}$:

$$\overrightarrow{\boldsymbol{x}} = (\overrightarrow{\boldsymbol{x}}^n)_{n=1:N} = ((b_{f,n}, \overline{p_f}, \overline{i_1}, \overline{i_2}, \overline{\tilde{w}_1}, \overline{\tilde{w}_2}, \overline{w_1}, \overline{w_2}))_{n \in \{1 \dots N\}}.$$

Future controls are approximated using cubic spline approximation.

B Proofs of the convergence speed

For a vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ we define the p-norm as $|x|_p = (x_1^p + \ldots + x_n^p)^{1/p}$. We denote the subordinate norm associated to $|\cdot|_p$ by $\|\cdot\|_p$. For a linear application $L : \mathbb{R}^n \to \mathbb{R}^n$ it is defined as

$$\|L\|_p = \sup_{x \neq 0} \frac{|L \cdot x|_p}{|x|_p}$$

and for multilinear application $M : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ mapping (x, y) to $M \cdot [x, y]$ we have:

$$\|M\|_p = \sup_{x \neq 0, y \neq 0} \frac{|M \cdot [x, y]|_p}{|x|_p |y|_p}.$$

For any linear application L we also denote by $\rho(L)$ the spectral radius of L, i.e., the absolute value of the largest complex eigenvalue λ such that there exists $x \in (\mathbb{R}^n)^*$ satisfying $Lx = \lambda x$. For any p we have $\rho(L) \leq \|L\|_p$. More precisely, the Gelfand formula states that $\rho(L) = \lim_{p \to \infty} \|L\|_p$. As a result, for any ϵ there exists p^ϵ such that $\|L\|_{p_\epsilon} < \rho(L) - \epsilon$.

Consider an application $T : \mathbb{R}^n \to \mathbb{R}^n$, for instance the time-iteration algorithm for the discretized model described in Section 2.1. Assume that T is continuously differentiable and that there exists a fixed-point $\overline{x} \in \mathbb{R}^n$ such that $T(\overline{x}) = \overline{x}$. Denote the spectral radius at the steady-state by $\lambda = \rho(T'(\overline{x}))$ and assume $\lambda < 1$. According to the property stated in the last paragraph, we choose q such that $\|\overline{T}'\|_q < \lambda$.

Under those assumptions we characterize the convergence speed of the algorithms exposed in Sections 3.2 and 6. These proofs are provided here, for reference, as they follow from standard results in real analysis.

B.1 Time iteration

Let us choose $\epsilon > 0$ such that $\rho + \epsilon < 1$. Since $T'(x)$ is continuous, there exists $\eta > 0$ such that for any x satisfying $|x - \overline{x}| < \eta$ we have $|T'(x) - T'(\overline{x})|_p < \epsilon$. It implies that for any such x, $\|T'(x)\|_p < \rho + \epsilon$.

Take $x, y \in \mathbb{R}^n$ such that $|x - \overline{x}|_p < \eta$ and $|y - \overline{x}|_p < \eta$. We can write

$$T(x) = T(y) + \int_{\lambda=0}^{1} T'(y + \lambda(x - y)) \cdot (x - y)\, d\lambda$$

from which we get:

$$\begin{aligned} |T(x) - T(y)|_p &\leq \int_{\lambda=0}^{1} |T'(y + \lambda(x - y))|_p |x - y|_p \, d\lambda \\ &\leq \int_{\lambda=0}^{1} |\rho + \epsilon| |x - y|_p \, d\lambda \\ &\leq (\rho + \epsilon)|x - y|_p. \end{aligned}$$

Finally choose $\kappa > 0$ such that $\kappa < \eta(1 - (\rho + \epsilon))$.

Given x_0 such that $|x_0 - \overline{x}|_p < \kappa$. For $k > 0$, construct the iterates $x_k = T(x_{k-1})$. It is easy to check recursively that all iterates x_k satisfy $|x_k - \overline{x}|_p < \eta$. Indeed, for any k we have

$$|x_k - \overline{x}|_p \leq |x_k - x_{k-1}|_p + \ldots + |x_1 - x_0|_p \leq ((\rho + \epsilon)^k + \ldots + (\rho + \epsilon))|x_0 - \overline{x}|_p \leq \frac{\eta}{1 - (\rho + \epsilon)} \kappa < \eta.$$

We can now show that the sequence $(x_k)_k$ converges to $\overline{x}$ using the local contractivity of T' for the norm $|\cdot|_p$. For any k we have

$$\begin{aligned} |x_k - \overline{x}|_p &\leq |x_k - x_{k+1}|_p + \ldots + |x_{k+l} - x_{k+l+1}|_p + \ldots \\ &\leq |x_k - x_{k+1}|_p (1 + (\rho + \epsilon)^2 + (\rho + \epsilon)^3 + \ldots) \\ &\leq \eta(\rho + \epsilon)^k \frac{1}{1 - \rho + \epsilon} \to_{k \to \infty} 0. \end{aligned}$$

This establishes that convergence is geometric with decay $\rho + \epsilon$ for any ϵ. Since all norms are equivalent in finite spaces, this is true for any norm, not just the p-norm.

B.2 Newton Raphson

Let us consider $G : \mathbb{R}^n \to \mathbb{R}^n$. Assume G is twice continuously differentiable. Assume there exists $\overline{x} \in \mathbb{R}^n$ such that $G(\overline{x}) = 0$ and that $G'(\overline{x})$ is invertible. From the inverse function theorem it follows that $(G'(\overline{x}))^{-1}$ is well defined and twice continuously differentiable in a neighborhood of $\overline{x}$.

We consider the function $\mathbb{R}^n \to \mathbb{R}^n$:

$$N(x) = x - (G'(x))^{-1}G(x)$$

and for a given initial guess x_0 and $k > 1$ define the Newton iterates as $x_k = N(x_{k-1}) = N^k(x_0)$. We prove below that sequence $(x_k)_k$ converges to $\overline{x}$ if x_0 is close enough to $\overline{x}$.

It is easy to check that $N(\overline{x}) = \overline{x}$ and that N is continuously differentiable. To compute the derivatives of N we write:

$$G'(x) \cdot (N(x) - x) + G(x) = 0.$$

Denoting by $B \cdot [u, v]$ the application of a bilinear operator to B to arguments u and v, we differentiate the last formula to get for any $u \in \mathbb{R}^n$

$$G''(x) \cdot [(N(x) - x), u] + G'(x) \cdot (N'(x) \cdot u - u) + G'(x) \cdot u = 0$$

and

$$\begin{aligned} N'(x) \cdot u &= u - (G'(x))^{-1}(G'(x) \cdot u + G''(x) \cdot [(N(x) - x), u]) \\ &= -(G'(x))^{-1}G''(x) \cdot [(N(x) - x), u]. \end{aligned} \tag{23}$$

Since $(G'(x))^{-1}G''(x)$ is continuous in x, its p-norm must be locally bounded by a positive real number C_1. From equation (23), we get in a neighborhood of $\overline{x}$:

$$\|N'(x)\|_p < C_1 \|N(x) - x\|_p.$$

In particular, this implies that $\rho(N'(\overline{x})) = 0$ and that the Newton iterates converge to $\overline{x}$ when x_0 is close enough to $\overline{x}$.

Since N is continuously differentiable and has $\overline{x}$ as a fixed point, there exists C_2 such that, close to $\overline{x}$

$$\begin{aligned} |x_{k+1} - x_k|_p = |N(x_k) - N(x_{k-1})|_p &\leq \|N'(x_{k-1})\|_p |x_k - x_{k-1}|_p + C_2(|x_k - x_{k-1}|_p)^2 \\ &\leq C_1 |x_k - x_{k-1}|_p^2 + C_2 |x_k - x_{k-1}|_p^2. \end{aligned}$$

Taking $C = \max(C_1, C_2)$ we thus have $|x_{k+1} - x_k|_p \leq C|x_k - x_{k-1}|_p^2$. This establishes that the convergence is *quadratic* in a neighborhood of $\overline{x}$.

B.3 Accelerated Time Iteration

Let us define the accelerated time-iteration operator as:

$$A(x) = x + (I - T'(x))^{-1}(T(x) - x).$$

Assuming T is twice continuously differentiable, let us define $G_T(x) = T(x) - x$. Then G_T is twice continuously differentiable too and we have $G_T'(x) = T'(x) - I$.

Since $\rho(T'(\overline{x})) < 1$, the application G_T' is invertible at $\overline{x}$ and by continuity in a neighborhood of $\overline{x}$. The equation

$$A(x) = x + (G_T'(x))^{-1}(G_T(x))$$

makes it obvious that application $A(\,\cdot\,)$ is actually a Newton step applied to G_T.

We can then use the result from the last subsection to conclude that iterates of A (and of G_T) converge at a *quadratic* rate.

C Detailed Timings

C.1 Model CS

C.1.1 Time Iterations

	Time			Allocations		
Tot / % measured:	102ms /	99.1%		78.6KiB /	1.2%	
Section	ncalls	time	%tot	avg	alloc	%tot	avg
solve for T	116	88.3ms	87.2%	761µs	976B	100.0%	8.41B
eval F()	498	55.0ms	54.3%	111µs	0.00B	0.0%	0.00B
compute F_1	194	31.7ms	31.3%	164µs	0.00B	0.0%	0.00B
eval F()	117	12.9ms	12.8%	111µs	0.00B	0.0%	0.00B
fit ϕ	117	35.7µs	0.0%	305ns	0.00B	0.0%	0.00B

C.1.2 Accelerated Time Iterations

	Time			Allocations		
Tot / % measured:	50.5ms /	98.4%		131KiB /	2.7%	
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	38.2ms	77.0%	7.65ms	1.31KiB	37.3%	269B
neumann sum	5	38.2ms	77.0%	7.65ms	672B	18.7%	134B
eval L	731	37.8ms	76.1%	51.7µs	0.00B	0.0%	0.00B
solve for T	5	8.30ms	16.7%	1.66ms	976B	27.1%	195B
eval F()	44	4.95ms	10.0%	112µs	0.00B	0.0%	0.00B
compute F_1	19	3.18ms	6.4%	168µs	0.00B	0.0%	0.00B
compute jacobian	5	2.47ms	5.0%	493µs	1.25KiB	35.6%	256B
compute F_2	5	1.49ms	3.0%	298µs	0.00B	0.0%	0.00B
compute F_1	5	835µs	1.7%	167µs	0.00B	0.0%	0.00B
compute L=-F_1\F_2	5	137µs	0.3%	27.4µs	0.00B	0.0%	0.00B
eval F()	6	677µs	1.4%	113µs	0.00B	0.0%	0.00B
fit ϕ	6	3.84µs	0.0%	639ns	0.00B	0.0%	0.00B

C.1.3 Accelerated Time Iterations (optimistic Q=50)

		Time			Allocations		
Tot / % measured:		25.3ms / 96.9%			131KiB / 2.7%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	13.0ms	53.2%	2.60ms	1.31KiB	37.3%	269B
neumann sum	5	13.0ms	53.2%	2.60ms	672B	18.7%	134B
eval L	250	12.9ms	52.6%	51.6µs	0.00B	0.0%	0.00B
solve for T	5	8.31ms	33.9%	1.66ms	976B	27.1%	195B
eval F()	44	4.95ms	20.2%	113µs	0.00B	0.0%	0.00B
compute F_1	19	3.20ms	13.1%	168µs	0.00B	0.0%	0.00B
compute jacobian	5	2.47ms	10.1%	494µs	1.25KiB	35.6%	256B
compute F_2	5	1.49ms	6.1%	298µs	0.00B	0.0%	0.00B
compute F_1	5	837µs	3.4%	167µs	0.00B	0.0%	0.00B
compute L=-F_1\F_2	5	137µs	0.6%	27.4µs	0.00B	0.0%	0.00B
eval F()	6	679µs	2.8%	113µs	0.00B	0.0%	0.00B
fit ϕ	6	3.15µs	0.0%	526ns	0.00B	0.0%	0.00B

C.1.4 Accelerated Time Iterations (GMRES Q=50)

		Time			Allocations		
Tot / % measured:		26.3ms / 96.9%			777KiB / 14.2%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	14.0ms	55.0%	2.80ms	107KiB	97.2%	21.4KiB
GMRES	5	14.0ms	54.8%	2.79ms	1.44KiB	1.3%	294B
compute G'.du	226	11.9ms	46.5%	52.5µs	0.00B	0.0%	0.00B
solve for T	5	8.32ms	32.7%	1.66ms	1.89KiB	1.7%	387B
eval F()	44	4.96ms	19.5%	113µs	0.00B	0.0%	0.00B
compute F_1	19	3.20ms	12.5%	168µs	0.00B	0.0%	0.00B
compute jacobian	5	2.47ms	9.7%	494µs	1.25KiB	1.1%	256B
compute F_2	5	1.49ms	5.9%	298µs	0.00B	0.0%	0.00B
compute F_1	5	838µs	3.3%	168µs	0.00B	0.0%	0.00B
compute L=-F_1\F_2	5	137µs	0.5%	27.4µs	0.00B	0.0%	0.00B
eval F()	6	679µs	2.7%	113µs	0.00B	0.0%	0.00B
fit ϕ	6	3.72µs	0.0%	620ns	0.00B	0.0%	0.00B

C.1.5 Newton–Krylov

		Time			Allocations		
Tot / % measured:		72.4ms / 89.1%			112KiB / 2.6%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	9	58.9ms	91.3%	6.54ms	1.31KiB	45.9%	149B
neumann sum	9	58.9ms	91.3%	6.54ms	672B	23.0%	74.7B
eval L	1.12k	58.2ms	90.3%	52.0µs	0.00B	0.0%	0.00B
compute Jacobian	9	4.50ms	7.0%	500µs	1.55KiB	54.1%	176B
compute F_2	9	2.64ms	4.1%	294µs	0.00B	0.0%	0.00B
compute F_1	9	1.50ms	2.3%	167µs	0.00B	0.0%	0.00B
compute L=-F_1\F_2	9	288µs	0.4%	32.0µs	0.00B	0.0%	0.00B
compute F_1\r0	9	59.7µs	0.1%	6.63µs	0.00B	0.0%	0.00B
eval F()	10	1.13ms	1.7%	113µs	0.00B	0.0%	0.00B
fit ϕ	10	4.61µs	0.0%	461ns	0.00B	0.0%	0.00B

C.1.6 Newton–Krylov (optimistic Q=50)

		Time			Allocations		
Tot / % measured:		26.7ms / 97.0%			79.2KiB / 3.6%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	8	20.9ms	80.6%	2.61ms	1.31KiB	45.9%	168B
neumann sum	8	20.9ms	80.6%	2.61ms	672B	23.0%	84.0B
eval L	400	20.6ms	79.7%	51.6µs	0.00B	0.0%	0.00B
compute Jacobian	8	3.99ms	15.4%	499µs	1.55KiB	54.1%	198B
compute F_2	8	2.36ms	9.1%	295µs	0.00B	0.0%	0.00B
compute F_1	8	1.35ms	5.2%	169µs	0.00B	0.0%	0.00B
compute L=-F_1\F_2	8	222µs	0.9%	27.7µs	0.00B	0.0%	0.00B
compute F_1\r0	8	53.2µs	0.2%	6.65µs	0.00B	0.0%	0.00B
eval F()	9	1.02ms	3.9%	113µs	0.00B	0.0%	0.00B
fit ϕ	9	5.05µs	0.0%	561ns	0.00B	0.0%	0.00B

C.1.7 Newton–Krylov (GMRES Q=50)

		Time			Allocations		
Tot / % measured:		32.9ms / 97.6%			271KiB / 71.4%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	9	26.6ms	82.6%	2.95ms	192KiB	99.2%	21.3KiB
GMRES	9	26.5ms	82.4%	2.94ms	2.06KiB	1.1%	235B
compute G'.du	429	22.4ms	69.6%	52.2µs	0.00B	0.0%	0.00B
compute Jacobian	9	4.46ms	13.9%	496µs	1.55KiB	0.8%	176B
compute F_2	9	2.65ms	8.2%	294µs	0.00B	0.0%	0.00B
compute F_1	9	1.50ms	4.7%	167µs	0.00B	0.0%	0.00B
compute L=-F_1\F_2	9	245µs	0.8%	27.2µs	0.00B	0.0%	0.00B
compute F_1\r0	9	59.7µs	0.2%	6.63µs	0.00B	0.0%	0.00B
eval F()	10	1.13ms	3.5%	113µs	0.00B	0.0%	0.00B
fit ϕ	10	5.27µs	0.0%	527ns	0.00B	0.0%	0.00B

C.1.8 Newton–Krylov (safeguarded, GMRES Q=50, no preconditioning)

		Time			Allocations		
Tot / % measured:		30.9ms / 97.1%			4.02MiB / 98.1%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
invert system	8	25.3ms	84.2%	3.16ms	3.86MiB	98.1%	494KiB
GMRES	8	25.2ms	84.0%	3.15ms	3.77MiB	95.6%	482KiB
compute G'.du	400	21.1ms	70.5%	52.9µs	3.69MiB	93.7%	9.45KiB
compute Jacobian	8	3.69ms	12.3%	462µs	976B	0.0%	122B
compute F_2	8	2.35ms	7.8%	294µs	0.00B	0.0%	0.00B
compute F_1	8	1.34ms	4.5%	167µs	0.00B	0.0%	0.00B
safeguard	8	940µs	3.1%	118µs	76.8KiB	1.9%	9.60KiB
eval F()	8	906µs	3.0%	113µs	0.00B	0.0%	0.00B
fit ϕ	8	4.12µs	0.0%	515ns	0.00B	0.0%	0.00B
eval F()	1	113µs	0.4%	113µs	0.00B	0.0%	0.00B
fit ϕ	1	788ns	0.0%	788ns	0.00B	0.0%	0.00B

C.1.9 Newton–Krylov (safeguarded, GMRES Q=50)

		Time			Allocations		
Tot / % measured:		33.0ms / 97.7%			358KiB / 78.2%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	9	26.6ms	82.5%	2.95ms	192KiB	68.6%	21.3KiB
GMRES	9	26.5ms	82.3%	2.95ms	2.06KiB	0.7%	235B
compute G'.du	429	22.4ms	69.5%	52.2µs	0.00B	0.0%	0.00B
compute Jacobian	9	4.47ms	13.9%	497µs	1.55KiB	0.6%	176B
compute F_2	9	2.65ms	8.2%	295µs	0.00B	0.0%	0.00B
compute F_1	9	1.51ms	4.7%	168µs	0.00B	0.0%	0.00B
compute L=-F_1\F_2	9	246µs	0.8%	27.3µs	0.00B	0.0%	0.00B
compute F_1\r0	9	59.7µs	0.2%	6.63µs	0.00B	0.0%	0.00B
safeguard	9	1.05ms	3.3%	117µs	86.2KiB	30.8%	9.58KiB
eval F()	9	1.01ms	3.1%	113µs	0.00B	0.0%	0.00B
fit ϕ	9	5.55µs	0.0%	617ns	0.00B	0.0%	0.00B
eval F()	1	113µs	0.3%	113µs	0.00B	0.0%	0.00B
fit ϕ	1	769ns	0.0%	769ns	0.00B	0.0%	0.00B

C.1.10 Newton–Krylov (safeguarded, GMRES Q=25)

		Time			Allocations		
Tot / % measured:		17.6ms / 95.9%			327KiB / 76.2%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	8	11.8ms	70.2%	1.48ms	171KiB	68.6%	21.4KiB
GMRES	8	11.8ms	70.0%	1.47ms	1.91KiB	0.8%	244B
compute G'.du	200	10.4ms	61.8%	52.0µs	0.00B	0.0%	0.00B
compute Jacobian	8	3.97ms	23.6%	496µs	1.55KiB	0.6%	198B
compute F_2	8	2.36ms	14.0%	294µs	0.00B	0.0%	0.00B
compute F_1	8	1.34ms	7.9%	167µs	0.00B	0.0%	0.00B
compute L=-F_1\F_2	8	218µs	1.3%	27.3µs	0.00B	0.0%	0.00B
compute F_1\r0	8	53.0µs	0.3%	6.62µs	0.00B	0.0%	0.00B
safeguard	8	929µs	5.5%	116µs	76.8KiB	30.8%	9.60KiB
eval F()	8	899µs	5.3%	112µs	0.00B	0.0%	0.00B
fit ϕ	8	3.25µs	0.0%	406ns	0.00B	0.0%	0.00B
eval F()	1	113µs	0.7%	113µs	0.00B	0.0%	0.00B
fit ϕ	1	746ns	0.0%	746ns	0.00B	0.0%	0.00B

C.2 Model RBC

C.2.1 Refinement : Iniital Time Iterations (T=25)

	Time			Allocations		
Tot / % measured:	277ms / 59.2%			7.89MiB / 10.5%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
solve for T	25	149ms	90.6%	5.94ms	739KiB	87.1%	29.6KiB
eval F()	125	75.8ms	46.2%	606µs	527KiB	62.1%	4.22KiB
compute F_1	50	57.2ms	34.9%	1.14ms	211KiB	24.8%	4.22KiB
eval F()	26	15.2ms	9.3%	584µs	110KiB	12.9%	4.22KiB
fit ϕ	26	162µs	0.1%	6.23µs	0.00B	0.0%	0.00B

C.2.2 Time Iterations

	Time			Allocations		
Tot / % measured:	820ms / 99.0%			4.24MiB / 98.2%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
solve for T	164	713ms	87.8%	4.35ms	3.48MiB	83.7%	21.7KiB
eval F()	618	379ms	46.6%	613µs	2.55MiB	61.2%	4.22KiB
compute F_1	227	262ms	32.2%	1.15ms	958KiB	22.5%	4.22KiB
eval F()	165	98.1ms	12.1%	594µs	696KiB	16.3%	4.22KiB
fit ϕ	165	1.09ms	0.1%	6.59µs	0.00B	0.0%	0.00B

C.2.3 Accelerated Time Iterations

	Time			Allocations		
Tot / % measured:	361ms / 99.7%			36.8MiB / 98.5%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	302ms	83.7%	60.3ms	2.61MiB	7.2%	534KiB
neumann sum	5	302ms	83.7%	60.3ms	2.61MiB	7.2%	534KiB
eval L	1.17k	273ms	75.7%	233µs	2.61MiB	7.2%	2.28KiB
solve for T	5	29.9ms	8.3%	5.99ms	149KiB	0.4%	29.7KiB
eval F()	25	15.1ms	4.2%	605µs	105KiB	0.3%	4.22KiB
compute F_1	10	11.5ms	3.2%	1.15ms	42.2KiB	0.1%	4.22KiB
compute jacobian	5	25.1ms	7.0%	5.03ms	33.4MiB	92.3%	6.69MiB
compute L=-F_1\F_2	5	12.0ms	3.3%	2.39ms	33.4MiB	92.2%	6.68MiB
compute F_2	5	7.45ms	2.1%	1.49ms	24.2KiB	0.1%	4.84KiB
compute F_1	5	5.73ms	1.6%	1.15ms	21.1KiB	0.1%	4.22KiB
eval F()	6	3.49ms	1.0%	581µs	25.3KiB	0.1%	4.22KiB
fit ϕ	6	37.7µs	0.0%	6.29µs	0.00B	0.0%	0.00B

C.2.4 Accelerated Time Iterations (optimistic Q=50)

		Time			Allocations		
Tot / % measured:		117ms / 99.0%			34.7MiB / 98.4%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	66.5ms	57.3%	13.3ms	574KiB	1.6%	115KiB
neumann sum	5	66.5ms	57.3%	13.3ms	573KiB	1.6%	115KiB
eval L	250	60.3ms	52.0%	241µs	570KiB	1.6%	2.28KiB
solve for T	5	27.0ms	23.3%	5.40ms	136KiB	0.4%	27.2KiB
eval F()	23	13.8ms	11.9%	601µs	97.0KiB	0.3%	4.22KiB
compute F_1	9	10.3ms	8.9%	1.15ms	38.0KiB	0.1%	4.22KiB
compute jacobian	5	19.1ms	16.5%	3.82ms	33.4MiB	97.9%	6.69MiB
compute F_2	5	7.44ms	6.4%	1.49ms	24.2KiB	0.1%	4.84KiB
compute L=-F_1\F_2	5	5.93ms	5.1%	1.19ms	33.4MiB	97.8%	6.68MiB
compute F_1	5	5.73ms	4.9%	1.15ms	21.1KiB	0.1%	4.22KiB
eval F()	6	3.43ms	3.0%	572µs	25.3KiB	0.1%	4.22KiB
fit ϕ	6	40.4µs	0.0%	6.73µs	0.00B	0.0%	0.00B

C.2.5 Accelerated Time Iterations (GMRES Q=50)

		Time			Allocations		
Tot / % measured:		143ms / 98.3%			40.8MiB / 85.7%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	78.6ms	55.8%	15.7ms	1.37MiB	3.9%	280KiB
GMRES	5	78.4ms	55.7%	15.7ms	412KiB	1.1%	82.3KiB
compute G'.du	170	52.1ms	37.0%	307µs	398KiB	1.1%	2.34KiB
solve for T	5	34.9ms	24.7%	6.97ms	149KiB	0.4%	29.8KiB
eval F()	25	18.2ms	12.9%	730µs	105KiB	0.3%	4.22KiB
compute F_1	10	13.4ms	9.5%	1.34ms	42.2KiB	0.1%	4.22KiB
compute jacobian	5	23.5ms	16.7%	4.71ms	33.4MiB	95.6%	6.69MiB
compute F_2	5	9.22ms	6.5%	1.84ms	24.2KiB	0.1%	4.84KiB
compute L=-F_1\F_2	5	7.46ms	5.3%	1.49ms	33.4MiB	95.5%	6.68MiB
compute F_1	5	6.85ms	4.9%	1.37ms	21.1KiB	0.1%	4.22KiB
eval F()	6	3.88ms	2.8%	647µs	25.3KiB	0.1%	4.22KiB
fit ϕ	6	44.9µs	0.0%	7.49µs	0.00B	0.0%	0.00B

C.2.6 Newton–Krylov

		Time			Allocations		
Tot / % measured:		411ms / 94.7%			36.3MiB / 99.5%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	304ms	78.2%	60.8ms	2.64MiB	7.3%	540KiB
neumann sum	5	304ms	78.2%	60.8ms	2.64MiB	7.3%	540KiB
eval L	1.18k	275ms	70.8%	233µs	2.63MiB	7.3%	2.28KiB
compute Jacobian	5	81.3ms	20.9%	16.3ms	33.4MiB	92.6%	6.69MiB
compute L=-F_1\F_2	5	66.8ms	17.2%	13.4ms	33.4MiB	92.5%	6.68MiB
compute F_2	5	7.42ms	1.9%	1.48ms	24.2KiB	0.1%	4.84KiB
compute F_1	5	5.76ms	1.5%	1.15ms	21.1KiB	0.1%	4.22KiB
compute F_1\r0	5	1.28ms	0.3%	256µs	0.00B	0.0%	0.00B
eval F()	6	3.43ms	0.9%	572µs	25.3KiB	0.1%	4.22KiB
fit ϕ	6	45.2µs	0.0%	7.54µs	0.00B	0.0%	0.00B

C.2.7 Newton–Krylov (optimistic Q=50)

		Time			Allocations		
Tot / % measured:		91.2ms / 98.9%			34.1MiB / 99.8%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	64.8ms	71.8%	13.0ms	574KiB	1.6%	115KiB
neumann sum	5	64.8ms	71.8%	13.0ms	573KiB	1.6%	115KiB
eval L	250	58.6ms	65.0%	234µs	570KiB	1.6%	2.28KiB
compute Jacobian	5	21.9ms	24.3%	4.38ms	33.4MiB	98.3%	6.69MiB
compute F_2	5	7.51ms	8.3%	1.50ms	24.2KiB	0.1%	4.84KiB
compute L=-F_1\F_2	5	7.21ms	8.0%	1.44ms	33.4MiB	98.1%	6.68MiB
compute F_1	5	5.86ms	6.5%	1.17ms	21.1KiB	0.1%	4.22KiB
compute F_1\r0	5	1.29ms	1.4%	258µs	0.00B	0.0%	0.00B
eval F()	6	3.48ms	3.9%	580µs	25.3KiB	0.1%	4.22KiB
fit ϕ	6	38.4µs	0.0%	6.40µs	0.00B	0.0%	0.00B

C.2.8 Newton–Krylov (GMRES Q=50)

		Time			Allocations		
Tot / % measured:		127ms / 99.1%			35.0MiB / 99.8%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	92.0ms	73.2%	18.4ms	1.42MiB	4.1%	292KiB
GMRES	5	91.8ms	73.0%	18.4ms	470KiB	1.3%	94.0KiB
compute G'.du	195	59.6ms	47.4%	305µs	457KiB	1.3%	2.34KiB
compute Jacobian	5	28.6ms	22.7%	5.71ms	33.4MiB	95.8%	6.69MiB
compute L=-F_1\F_2	5	12.3ms	9.8%	2.46ms	33.4MiB	95.7%	6.68MiB
compute F_2	5	7.98ms	6.3%	1.60ms	24.2KiB	0.1%	4.84KiB
compute F_1	5	6.96ms	5.5%	1.39ms	21.1KiB	0.1%	4.22KiB
compute F_1\r0	5	1.29ms	1.0%	257µs	0.00B	0.0%	0.00B
eval F()	6	5.15ms	4.1%	859µs	25.3KiB	0.1%	4.22KiB
fit ϕ	6	41.7µs	0.0%	6.96µs	0.00B	0.0%	0.00B

C.2.9 Newton–Krylov (safeguarded, GMRES Q=50, no preconditioning)

		Time			Allocations		
Tot / % measured:		1.94s / 99.7%			172MiB / 99.9%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
invert system	30	1.67s	86.3%	55.7ms	149MiB	86.9%	4.97MiB
GMRES	30	1.67s	86.2%	55.7ms	146MiB	85.2%	4.87MiB
compute G'.du	1.50k	1.36s	70.4%	910µs	143MiB	83.5%	97.8KiB
safeguard	30	169ms	8.7%	5.63ms	22.2MiB	13.0%	758KiB
eval F()	223	157ms	8.1%	702µs	941KiB	0.5%	4.22KiB
fit ϕ	223	1.50ms	0.1%	6.72µs	0.00B	0.0%	0.00B
compute Jacobian	30	95.2ms	4.9%	3.17ms	273KiB	0.2%	9.09KiB
compute F_2	30	54.3ms	2.8%	1.81ms	145KiB	0.1%	4.84KiB
compute F_1	30	40.8ms	2.1%	1.36ms	127KiB	0.1%	4.22KiB
eval F()	1	1.17ms	0.1%	1.17ms	4.22KiB	0.0%	4.22KiB
fit ϕ	1	9.69µs	0.0%	9.69µs	0.00B	0.0%	0.00B

C.2.10 Newton–Krylov (safeguarded, GMRES Q=50)

		Time			Allocations		
Tot / % measured:		446ms / 99.8%			65.4MiB / 99.9%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	9	347ms	78.1%	38.6ms	2.71MiB	4.1%	308KiB
GMRES	9	347ms	78.0%	38.6ms	0.97MiB	1.5%	110KiB
compute G'.du	414	264ms	59.3%	637µs	970KiB	1.5%	2.34KiB
compute Jacobian	9	66.8ms	15.0%	7.42ms	60.2MiB	92.2%	6.69MiB
compute L=-F_1\F_2	9	27.0ms	6.1%	3.00ms	60.1MiB	92.1%	6.68MiB
compute F_2	9	20.7ms	4.7%	2.30ms	43.6KiB	0.1%	4.84KiB
compute F_1	9	16.7ms	3.8%	1.86ms	38.0KiB	0.1%	4.22KiB
compute F_1\r0	9	2.34ms	0.5%	260µs	0.00B	0.0%	0.00B
safeguard	9	29.2ms	6.6%	3.25ms	2.39MiB	3.7%	272KiB
eval F()	24	27.9ms	6.3%	1.16ms	101KiB	0.2%	4.22KiB
fit ϕ	24	163µs	0.0%	6.78µs	0.00B	0.0%	0.00B
eval F()	1	1.21ms	0.3%	1.21ms	4.22KiB	0.0%	4.22KiB
fit ϕ	1	8.34µs	0.0%	8.34µs	0.00B	0.0%	0.00B

C.2.11 Newton–Krylov (safeguarded, GMRES Q=25)

		Time			Allocations		
Tot / % measured:		242ms / 99.6%			65.0MiB / 99.9%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	9	165ms	68.5%	18.4ms	2.28MiB	3.5%	259KiB
GMRES	9	165ms	68.4%	18.3ms	550KiB	0.8%	61.2KiB
compute G'.du	225	137ms	56.9%	610µs	527KiB	0.8%	2.34KiB
compute Jacobian	9	53.5ms	22.2%	5.94ms	60.2MiB	92.8%	6.69MiB
compute F_2	9	18.3ms	7.6%	2.03ms	43.6KiB	0.1%	4.84KiB
compute L=-F_1\F_2	9	17.5ms	7.3%	1.95ms	60.1MiB	92.7%	6.68MiB
compute F_1	9	15.3ms	6.3%	1.70ms	38.0KiB	0.1%	4.22KiB
compute F_1\r0	9	2.34ms	1.0%	260µs	0.00B	0.0%	0.00B
safeguard	9	21.2ms	8.8%	2.36ms	2.39MiB	3.7%	272KiB
eval F()	24	20.1ms	8.3%	837µs	101KiB	0.2%	4.22KiB
fit ϕ	24	154µs	0.1%	6.40µs	0.00B	0.0%	0.00B
eval F()	1	1.18ms	0.5%	1.18ms	4.22KiB	0.0%	4.22KiB
fit ϕ	1	8.12µs	0.0%	8.12µs	0.00B	0.0%	0.00B

C.3 Model FI

C.3.1 Refinement : Iniital Time Iterations (T=5)

		Time			Allocations		
Tot / % measured:		187s / 99.8%			36.3MiB / 80.3%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
solve for T	5	177s	95.1%	35.4s	1.59MiB	5.5%	326KiB
compute F_1	18	108s	57.7%	5.98s	497KiB	1.7%	27.6KiB
eval F()	41	62.4s	33.5%	1.52s	1.11MiB	3.8%	27.6KiB
eval F()	6	8.99s	4.8%	1.50s	166KiB	0.6%	27.6KiB
fit ϕ	6	192ms	0.1%	31.9ms	27.4MiB	94.0%	4.57MiB

C.3.2 Time Iterations

		Time			Allocations		
Tot / % measured:		1.52h / 99.8%			1.53GiB / 99.8%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
solve for T	330	1.37h	90.3%	14.9s	46.0MiB	2.9%	143KiB
compute F_1	458	2804s	51.5%	6.12s	12.4MiB	0.8%	27.6KiB
eval F()	1.25k	1931s	35.4%	1.55s	33.6MiB	2.1%	27.6KiB
eval F()	331	517s	9.5%	1.56s	8.93MiB	0.6%	27.6KiB
fit ϕ	331	10.4s	0.2%	31.5ms	1.48GiB	96.5%	4.57MiB

C.3.3 Accelerated Time Iterations

		Time			Allocations		
Tot / % measured:		1762s / 100.0%			78.2GiB / 99.6%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	1572s	89.2%	314s	7.68GiB	9.9%	1.54GiB
neumann sum	5	1572s	89.2%	314s	7.68GiB	9.9%	1.54GiB
eval L	1.71k	1529s	86.8%	892ms	7.67GiB	9.9%	4.59MiB
solve for T	5	113s	6.4%	22.7s	1.03MiB	0.0%	210KiB
compute F_1	11	67.3s	3.8%	6.12s	304KiB	0.0%	27.6KiB
eval F()	27	41.8s	2.4%	1.55s	746KiB	0.0%	27.6KiB
compute jacobian	5	67.4s	3.8%	13.5s	70.2GiB	90.1%	14.0GiB
compute F_1	5	30.6s	1.7%	6.11s	138KiB	0.0%	27.6KiB
compute F_2	5	19.1s	1.1%	3.83s	178KiB	0.0%	35.6KiB
compute L=-F_1\F_2	5	17.7s	1.0%	3.54s	70.2GiB	90.1%	14.0GiB
eval F()	6	9.08s	0.5%	1.51s	166KiB	0.0%	27.6KiB
fit ϕ	6	307ms	0.0%	51.2ms	27.4MiB	0.0%	4.57MiB

C.3.4 Accelerated Time Iterations (optimistic Q=50)

		Time			Allocations		
Tot / % measured:		623s / 99.9%			115GiB / 99.5%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	8	358s	57.6%	44.8s	1.79GiB	1.6%	229MiB
neumann sum	8	358s	57.6%	44.8s	1.79GiB	1.6%	229MiB
eval L	400	348s	56.0%	870ms	1.79GiB	1.6%	4.58MiB
solve for T	8	145s	23.3%	18.1s	1.35MiB	0.0%	173KiB
compute F_1	14	84.1s	13.5%	6.01s	387KiB	0.0%	27.6KiB
eval F()	36	55.0s	8.8%	1.53s	0.97MiB	0.0%	27.6KiB
compute jacobian	8	105s	16.9%	13.1s	112GiB	98.4%	14.0GiB
compute F_1	8	48.0s	7.7%	6.00s	221KiB	0.0%	27.6KiB
compute F_2	8	30.3s	4.9%	3.79s	285KiB	0.0%	35.6KiB
compute L=-F_1\F_2	8	26.6s	4.3%	3.32s	112GiB	98.4%	14.0GiB
eval F()	9	13.6s	2.2%	1.51s	249KiB	0.0%	27.6KiB
fit ϕ	9	284ms	0.0%	31.5ms	41.1MiB	0.0%	4.57MiB

C.3.5 Accelerated Time Iterations (GMRES Q=50)

		Time			Allocations		
Tot / % measured:		491s / 99.7%			76.0GiB / 94.7%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	304s	62.0%	60.7s	1.81GiB	2.5%	371MiB
GMRES	5	303s	61.9%	60.7s	1.14GiB	1.6%	234MiB
compute G'.du	250	226s	46.1%	902ms	1.12GiB	1.6%	4.58MiB
solve for T	5	111s	22.8%	22.3s	1.03MiB	0.0%	210KiB
compute F_1	11	66.1s	13.5%	6.01s	304KiB	0.0%	27.6KiB
eval F()	27	41.1s	8.4%	1.52s	746KiB	0.0%	27.6KiB
compute jacobian	5	65.7s	13.4%	13.1s	70.2GiB	97.4%	14.0GiB
compute F_1	5	30.1s	6.1%	6.01s	138KiB	0.0%	27.6KiB
compute F_2	5	18.9s	3.9%	3.78s	178KiB	0.0%	35.6KiB
compute L=-F_1\F_2	5	16.7s	3.4%	3.35s	70.2GiB	97.4%	14.0GiB
eval F()	6	8.99s	1.8%	1.50s	166KiB	0.0%	27.6KiB
fit ϕ	6	209ms	0.0%	34.8ms	27.4MiB	0.0%	4.57MiB

C.3.6 Newton–Krylov

		Time			Allocations		
Tot / % measured:		1506s / 100.0%			77.2GiB / 100.0%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	1428s	94.8%	286s	7.04GiB	9.1%	1.41GiB
neumann sum	5	1428s	94.8%	286s	7.04GiB	9.1%	1.41GiB
eval L	1.57k	1389s	92.2%	883ms	7.04GiB	9.1%	4.58MiB
compute Jacobian	5	68.7s	4.6%	13.7s	70.2GiB	90.8%	14.0GiB
compute F_1	5	30.5s	2.0%	6.09s	138KiB	0.0%	27.6KiB
compute F_2	5	19.2s	1.3%	3.83s	178KiB	0.0%	35.6KiB
compute L=-F_1\F_2	5	17.4s	1.2%	3.49s	70.2GiB	90.8%	14.0GiB
compute F_1\r0	5	1.65s	0.1%	330ms	0.00B	0.0%	0.00B
eval F()	6	9.15s	0.6%	1.52s	166KiB	0.0%	27.6KiB
fit ϕ	6	192ms	0.0%	32.0ms	27.4MiB	0.0%	4.57MiB

C.3.7 Newton–Krylov (optimistic Q=50)

		Time			Allocations		
Tot / % measured:		480s / 99.9%			114GiB / 100.0%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	8	358s	74.6%	44.7s	1.79GiB	1.6%	229MiB
neumann sum	8	358s	74.6%	44.7s	1.79GiB	1.6%	229MiB
eval L	400	347s	72.5%	869ms	1.79GiB	1.6%	4.58MiB
compute Jacobian	8	108s	22.5%	13.5s	112GiB	98.4%	14.0GiB
compute F_1	8	47.9s	10.0%	5.99s	221KiB	0.0%	27.6KiB
compute F_2	8	30.4s	6.3%	3.80s	285KiB	0.0%	35.6KiB
compute L=-F_1\F_2	8	27.0s	5.6%	3.37s	112GiB	98.4%	14.0GiB
compute F_1\r0	8	2.59s	0.5%	324ms	0.00B	0.0%	0.00B
eval F()	9	13.6s	2.8%	1.51s	249KiB	0.0%	27.6KiB
fit ϕ	9	284ms	0.1%	31.6ms	41.1MiB	0.0%	4.57MiB

C.3.8 Newton–Krylov (GMRES Q=50)

		Time			Allocations		
Tot / % measured:		380s / 99.9%			72.0GiB / 100.0%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	5	303s	79.8%	60.6s	1.81GiB	2.5%	371MiB
GMRES	5	303s	79.7%	60.6s	1.14GiB	1.6%	234MiB
compute G'.du	250	225s	59.3%	901ms	1.12GiB	1.6%	4.58MiB
compute Jacobian	5	67.7s	17.8%	13.5s	70.2GiB	97.4%	14.0GiB
compute F_1	5	30.0s	7.9%	6.01s	138KiB	0.0%	27.6KiB
compute F_2	5	19.1s	5.0%	3.81s	178KiB	0.0%	35.6KiB
compute L=-F_1\F_2	5	16.9s	4.5%	3.39s	70.2GiB	97.4%	14.0GiB
compute F_1\r0	5	1.64s	0.4%	327ms	0.00B	0.0%	0.00B
eval F()	6	9.00s	2.4%	1.50s	166KiB	0.0%	27.6KiB
fit ϕ	6	213ms	0.1%	35.5ms	27.4MiB	0.0%	4.57MiB

C.3.9 Newton–Krylov (safeguarded, GMRES Q=50)

		Time			Allocations		
Tot / % measured:		455s / 100.0%			86.9GiB / 100.0%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	6	363s	79.8%	60.6s	2.17GiB	2.5%	371MiB
GMRES	6	363s	79.8%	60.5s	1.37GiB	1.6%	234MiB
compute G'.du	300	270s	59.3%	900ms	1.34GiB	1.5%	4.58MiB
compute Jacobian	6	80.7s	17.7%	13.5s	84.2GiB	96.9%	14.0GiB
compute F_1	6	35.8s	7.9%	5.96s	166KiB	0.0%	27.6KiB
compute F_2	6	22.7s	5.0%	3.78s	214KiB	0.0%	35.6KiB
compute L=-F_1\F_2	6	20.3s	4.5%	3.39s	84.2GiB	96.9%	14.0GiB
compute F_1\r0	6	1.97s	0.4%	328ms	0.00B	0.0%	0.00B
safeguard	6	9.92s	2.2%	1.65s	513MiB	0.6%	85.5MiB
eval F()	7	9.24s	2.0%	1.32s	195KiB	0.0%	27.9KiB
fit ϕ	7	255ms	0.1%	36.5ms	32.0MiB	0.0%	4.57MiB
eval F()	1	1.20s	0.3%	1.20s	27.6KiB	0.0%	27.6KiB
fit ϕ	1	34.3ms	0.0%	34.3ms	4.57MiB	0.0%	4.57MiB

C.3.10 Newton–Krylov (safeguarded, GMRES Q=25)

		Time			Allocations		
Tot / % measured:		341s / 100.0%			115GiB / 100.0%		
Section	ncalls	time	%tot	avg	alloc	%tot	avg
compute (I-L)^(-1).dr	8	219s	64.1%	27.3s	2.00GiB	1.7%	257MiB
GMRES	8	218s	64.0%	27.3s	953MiB	0.8%	119MiB
compute G'.du	200	180s	52.8%	900ms	917MiB	0.8%	4.58MiB
compute Jacobian	8	108s	31.7%	13.5s	112GiB	97.7%	14.0GiB
compute F_1	8	47.9s	14.1%	5.99s	221KiB	0.0%	27.6KiB
compute F_2	8	30.2s	8.9%	3.77s	285KiB	0.0%	35.6KiB
compute L=-F_1\F_2	8	27.2s	8.0%	3.40s	112GiB	97.7%	14.0GiB
compute F_1\r0	8	2.61s	0.8%	326ms	0.00B	0.0%	0.00B
safeguard	8	13.1s	3.9%	1.64s	659MiB	0.6%	82.4MiB
eval F()	9	12.4s	3.6%	1.38s	251KiB	0.0%	27.9KiB
fit ϕ	9	341ms	0.1%	37.9ms	41.1MiB	0.0%	4.57MiB
eval F()	1	1.22s	0.4%	1.22s	27.6KiB	0.0%	27.6KiB
fit ϕ	1	29.6ms	0.0%	29.6ms	4.57MiB	0.0%	4.57MiB