Solving continuous time perfect-foresight models

Stéphane Adjemian

Université du Mans

June 19, 2026

Plan

Plan

Perfect foresight – What we are solving

- Macro/finance models routinely produce systems of differential or difference equations mixing **predetermined states** (capital, debt, habit stock) and **forward-looking jumps** (consumption, inflation, asset prices, costates).
- Under **perfect foresight**, agents know the entire future path of exogenous variables. The model collapses to a **deterministic two-point boundary-value problem (BVP)**:
  - states are pinned at $t = 0$ by their initial condition,
  - jumps are pinned far in the future by the terminal steady state (saddle-path stability).
- The BVP is non-linear, typically has no closed form, and must be solved **numerically**: discretise (if time is continuous), then run Newton.

Perfect foresight – Why bother?

- Perfect-foresight transitions are the workhorse for:
  - transition dynamics after a permanent policy change (Ramsey, fiscal reform),
  - the response to a pre-announced (anticipated) sequence of shocks,
  - occasionally-binding constraints (ZLB, borrowing limits) where local linearisation fails,
  - non-linear models where second-order Taylor expansions miss the action.

Perfect foresight – Two flavours

- **Discrete time** (Dynare's natural setting): the model is a system of non-linear equations

$$f\big(y_{t+1},\, y_t,\, y_{t-1},\, u_t,\, \theta\big) \;=\; 0, \qquad t = 1, \ldots, T,$$

with y_0 given (states) and $y_{T+1} = \bar{y}$ (terminal steady state).

- **Continuous time** (Continuo's setting): the model is an implicit differential-algebraic equation (DAE)

$$F\big(\dot{x}(t),\, x(t),\, e(t),\, \theta,\, t\big) \;=\; 0, \qquad t \in [0, T],$$

with $x_S(0)$ given (states) and $x_J(T) = \bar{x}_J$ (jumps at the terminal steady state).

- Both reduce, after suitable discretisation, to a large block-sparse **non-linear** system solved by Newton. The difference lies in *how* the time derivative is approximated.

Plan

Continuous time – The model

- The model is an implicit DAE on the horizon $[0, T]$:

$$\boxed{F\big(\dot{x},\, x,\, e,\, \theta,\, t\big) \;=\; 0,}$$

  $x \in \mathbb{R}^n$, $e \in \mathbb{R}^{n_e}$ exogenous, θ parameters. F takes values in $\mathbb{R}^n$.

- Endogenous variables split into n_S **states**, n_J **jumps** (together: $n_d = n_S + n_J$ *dynamic* variables, the ones with a time derivative), and $n_a = n - n_d$ **algebraic** variables, defined by static relations.

- Equations split accordingly into n_d *dynamic rows* (depending on $\dot{x}$) and n_a *algebraic rows* (pure equalities). Continuo detects the partition automatically from CasADi's symbolic dependency graph.

Continuous time – Boundary conditions

- **Initial condition (states).** Each state x_s ($s \in S$) is pinned at $t = 0$:

$$x_s(0) \;=\; x_s^0, \qquad s \in S.$$

  In Continuo these values come from the `initval` block (often the initial steady state).

- **Terminal condition (jumps).** Each jump x_j ($j \in J$) is pinned at the *terminal* steady state:

$$x_j(T) \;=\; x_j^\star, \qquad j \in J,$$

  computed at the exogenous belief $e(T)$.

- **Algebraic variables** need no boundary condition: their values are determined at every t by the algebraic rows of F.

- Together, $n_S + n_J = n_d$ boundary conditions, matching the n_d first-order differential equations. The BVP is well-posed.

Continuous time – Example: Ramsey–Cass–Koopmans

- Felicity $u(c) = c^{1-\sigma}/(1-\sigma)$, technology $f(k)$, depreciation δ, discount ρ, single exogenous TFP shifter z entering f:

$$\dot{k} \;=\; z\,f(k) - \delta k - c, \qquad \dot{c} \;=\; \tfrac{c}{\sigma}\big(z\,f'(k) - \delta - \rho\big).$$

- State: k (capital, predetermined). Jump: c (consumption, chosen at every t).

- Boundary conditions:

$$k(0) = k_0, \qquad c(T) = \bar{c}\big(z(T)\big).$$

- No algebraic variables here ($n_a = 0$). In general Continuo handles $n_a > 0$ transparently (intermediate definitions, market clearing, etc.).

Plan

Discretisation – Strategy

- Choose a grid on $[0, T]$ with N intervals:

$$0 = t_0 < t_1 < \cdots < t_N = T, \qquad dt = T/N, \quad t_{i+\frac{1}{2}} = \tfrac{1}{2}(t_i + t_{i+1}).$$

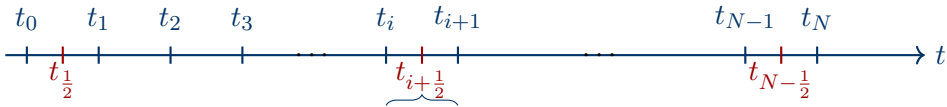
- Replace the BVP by $N+1$ *unknown vectors* $x_0, x_1, \ldots, x_N \in \mathbb{R}^n$, one per grid point, stacked into

$$X \;=\; \left(x_0^\top, x_1^\top, \ldots, x_N^\top\right)^\top \;\in\; \mathbb{R}^{n(N+1)}.$$

- Replace the differential equation by an **algebraic** link between consecutive points (*collocation*), plus enforce the algebraic rows pointwise and the boundary conditions.

- Result: a square non-linear system $G(X) = 0$ of size $n(N+1)$, solved by **Newton** with a sparse linear solver.

Discretisation – The grid



- $N+1$ **grid points** $t_0, \ldots, t_N$ (blue): the unknowns $x_i \equiv x(t_i)$ live here.
- N **midpoints** $t_{i+\frac{1}{2}}$ (red): additional collocation points used by some schemes to evaluate the model inside each interval.
- The default grid is uniform (dt the same everywhere), but the nodes need not be: Continuo aligns them to shock reveal times and can refine them **adaptively** where the solution is hard (see *Grids and numerical experiments*).

Plan

Approximation – The one-step problem

- A **one-step scheme** for $\dot{x} = g(x,t)$ produces x_{i+1} from x_i alone, possibly via internal evaluations of g inside $[t_i, t_{i+1}]$ but without referencing earlier $x_{i-1}, x_{i-2}, \ldots$ (as multistep methods would do).

- Such a scheme is judged by how well one step approximates the exact **flow map** $x(t_{i+1}) = \Phi_{dt}\big(x(t_i)\big)$: consistency, accuracy, stability all reduce to how close the scheme's update is to Φ_{dt}.

- As we will demonstrate below, Φ_{dt} can be constructed straightforwardly when the exact solution to the differential equation is available.

- We will focus on the first-order linear homogeneous ordinary differential equation.

Approximation – A simple test problem

- We consider the simplest ODE:
$$\dot{x} = \lambda\, x$$
with $\lambda \in \mathbb{C}$, $x(0) = 1$. The exact solution $x(t) = e^{\lambda t}$ converges to 0 iff $\mathrm{Re}(\lambda) < 0$.

- Every one-step scheme applied to the test problem produces an iterate of the form $x_{i+1} = A(\lambda\, dt)\, x_i$. The **amplification factor** $A(z)$, $z = \lambda\, dt$, is the scheme's approximation of e^z. Two fundamental questions:
  - *Accuracy.* How many Taylor coefficients of e^z does A get right?
  - *Stability.* Is $|A(z)| \leq 1$ on the left half-plane $\mathrm{Re}(z) < 0$?
- **Conditionally stable**: $|A| \leq 1$ only for dt below some λ-dependent threshold. **A-stable**: $|A| \leq 1$ for every z with $\mathrm{Re}(z) < 0$. **L-stable**: A-stable *and* $|A(z)| \to 0$ as $\mathrm{Re}(z) \to -\infty$ (cf. next slide).

Approximation – A-stable vs. L-stable

- A-stability prevents blow-up but says nothing about damping: for $|\lambda\, dt|$ very large $e^{\lambda dt} \approx 0$, yet $|A(\lambda dt)|$ may sit near 1 — and when $A \approx -1$ the iterate flips sign at every step ($+, -, +, -$ at amplitude ~ 1): a parasitic sawtooth, while the true solution decays monotonically. **L-stability** adds $|A(z)| \to 0$ as $|z| \to \infty$ on the left half-plane, killing fast modes in one step.

- Two representative rational $A(z)$ at $\lambda\, dt = -100$:

$A(z)$	type	$A(-100)$	after one step
$(1 + z/2)/(1 - z/2)$	A-stable, not L	$(1{-}50)/(1{+}50) \approx -0.96$	sawtooth at amplitude ~ 1
$1/(1 - z)$	L-stable	$1/(1{+}100) \approx 0.01$	damped to ~ 0
e^z (exact)	—	$\sim 10^{-44}$	essentially zero

- This matters in *multivariate* problems whose Jacobian has eigenvalues of widely different magnitudes — the standard notion of **stiffness**. Usually, A-stability suffices for perfect-foresight macro. L-stability is mentioned only because the Padé catalogue (next section) will classify some schemes as such.

Approximation – Local vs. global error

- **Local error** (per step). Assume $x_i = x(t_i)$ exactly, apply the scheme once: $e_i^{\text{loc}} = x_{i+1}^{\text{scheme}} - x(t_{i+1})$ — the error from a *single* step, starting from the truth.

- **Global error** at horizon T. Run the scheme from $t_0 = 0$ to $t_N = T$ with $N = T/dt$ steps: $e^{\text{glob}}(T) = x_N^{\text{scheme}} - x(T)$.

- **Order.** A scheme is of *global order* p if $e_i^{\text{loc}} = O(dt^{p+1})$. Summing over $N = T/dt$ steps,

$$e^{\text{glob}}(T) \;=\; \underbrace{O(T/dt)}_{N \sim 1/dt} \cdot O(dt^{p+1}) \;=\; T \cdot O\big(dt^{p+1}/dt\big) \;=\; O(dt^{p})$$

  the $1/dt$ (more steps) cancels exactly one power of dt. Local order = global order $+1$. When these slides say "scheme of order p", they mean *global* order p.

- **Caveat.** The stacking $N \times e^{\text{loc}}$ requires the scheme to be *stable*; otherwise local errors amplify exponentially.

Why is the simplest ODE of interest? – Nonlinear scalar problems

- For $\dot{x} = g(x,t)$, $x \in \mathbb{R}$, linearise around (x_i, t_i). With $\delta = x - x_i$, $\lambda_i = g'(x_i, t_i)$, $g_i = g(x_i, t_i)$ (constant over one step): $\dot{\delta} = \lambda_i\,\delta + g_i$, $\delta(0) = 0$.

- Multiply by $e^{-\lambda_i t}$ so $\frac{d}{dt}\left(e^{-\lambda_i t}\delta\right) = g_i\, e^{-\lambda_i t}$; integrate t from 0 to dt:

$$\delta(dt) \;=\; g_i\,\frac{e^{\lambda_i\,dt} - 1}{\lambda_i} \;=\; dt\,g_i + \tfrac{dt^2}{2}\lambda_i g_i + O(dt^3).$$

- **Forward Euler**, $x_{i+1} = x_i + dt\,g(x_i, t_i)$, on the nonlinear ODE reproduces only $dt\,g_i$ — local error $\frac{dt^2}{2}\lambda_i g_i = O(dt^2)$, hence **global order 1**. The same order emerges from the test problem $\dot{x} = \lambda x$: FE gives $x_{i+1} = x_i + \lambda\,x_i\,dt = (1 + \lambda\,dt)\,x_i$, defining its **amplification factor** $A_{\mathrm{FE}}(z) = 1 + z$ ($z = \lambda\,dt$). Compared to $e^z = 1 + z + z^2/2 + \cdots$, the gap is $A_{\mathrm{FE}}(z) - e^z = -z^2/2 + O(z^3)$ — local error $O(dt^2)$, global order 1, matching the nonlinear ODE. **General rule**: a scheme's global order p on any smooth ODE equals the order at which $A(z) - e^z = O(z^{p+1})$ at $z = 0$.

Why is the simplest ODE of interest? – Forced linear scalar problems

- For $\dot{x} = \mu x + u(t)$, $x \in \mathbb{R}$, with anticipated forcing $u(t)$, variation of parameters gives the exact one-step

$$x(t_{i+1}) \;=\; e^{\mu\, dt}\, x(t_i) \;+\; \int_{t_i}^{t_{i+1}} e^{\mu(t_{i+1}-s)}\, u(s)\, ds.$$

- Applying forward Euler:

$$x_{i+1} \;=\; x_i + dt\big(\mu x_i + u(t_i)\big) \;=\; \underbrace{(1+\mu\, dt)}_{A_{\mathrm{FE}}(\mu\, dt)}\, x_i \;+\; \underbrace{dt\, u(t_i)}_{\approx \int e^{\mu(\cdot)} u\, ds}\,.$$

  The $dt\, u(t_i)$ piece approximates the *full* kernel-weighted integral: for small dt both $e^{\mu(t_{i+1}-s)} \to 1$ and $u(s) \to u(t_i)$, leaving an $O(dt^2)$ remainder. Together with the $O(dt^2)$ homogeneous gap, FE is **global order 1**.

- General pattern: every one-step scheme gives $x_{i+1} = A(\mu\, dt)\, x_i +$ (weighted sum of u-samples). Net global order $= \min(p, q)$, with p from $A(z) - e^z = O(z^{p+1})$ and q the order of the scheme's *quadrature rule* on u.

Why is the simplest ODE of interest? – Nonlinear with forcing

- General scalar problem $\dot{x} = g(x, u(t), t)$. Linearise around (x_i, t_i) with $\delta = x - x_i$, $\lambda_i = \partial_x g$, $\beta_i = \partial_u g$, $g_i = g(x_i, u(t_i), t_i)$:

$$\dot{\delta} \;\approx\; \lambda_i\,\delta \;+\; g_i \;+\; \beta_i\,\big[u(t) - u(t_i)\big]$$

  — a linear ODE with constant drift g_i *and* time-varying forcing, combining the two preceding scenarios.

- Variation of parameters splits the exact one-step into three pieces: homogeneous propagator $e^{\lambda_i\,dt}$ (approximated by $A(z)$); constant-drift contribution $g_i\,(e^{\lambda_i\,dt}-1)/\lambda_i$ (reproduced by every consistent scheme via $dt\,g_i$); time-varying forcing integral (discretised by the scheme's quadrature on u).

- Net *global* order $= \min(p, q)$, where p comes from $A(z) - e^z = O(z^{p+1})$ on the homogeneous side and q is the order of the scheme's embedded *quadrature rule* on u ($q = 1$ for endpoint sampling, $q = 2$ for trapezoidal/midpoint, higher for richer rules). **No new analysis required.**

Padé – General principle

- For any function f analytic near 0 with Taylor series $f(z) = \sum_{k \geq 0} c_k \, z^k$, the **Padé approximant** $R_{p,q}$ of type (p, q) is the unique rational function $R_{p,q}(z) = P_p(z)/Q_q(z)$ with $\deg P_p \leq p$, $\deg Q_q \leq q$, $Q_q(0) = 1$, matching f to order $p + q$:

$$R_{p,q}(z) - f(z) \;=\; O\big(z^{p+q+1}\big).$$

- The $q = 0$ row recovers Taylor polynomials — Padé strictly generalises polynomial approximation.

- **Why rationals?** A rational function captures features no polynomial of finite degree can: *poles* (e.g. $1/(1 - z)$ blows up at $z = 1$) and *boundedness at infinity*. Same logic as transfer functions and rational kernels in time-series analysis.

- **When?** Whenever we want maximum information from a fixed amount of local Taylor data — convergence acceleration, special functions, model-order reduction, integrators for ODEs.

Padé – Construction and an illustration on $\log(1+z)$

- Multiplying out, $Q_q(z)\,f(z) - P_p(z) = O(z^{p+q+1})$. Matching coefficients of $z^0, \ldots, z^{p+q}$ gives a linear system: q equations (orders $p{+}1, \ldots, p{+}q$) determine Q_q uniquely, then $p{+}1$ equations cascade to give P_p.

- **Illustration:** $f(z) = \log(1+z)$, $c_k = (-1)^{k-1}/k$:

$$R_{1,0}(z) = z, \qquad R_{1,1}(z) = \frac{z}{1 + z/2}, \qquad R_{2,2}(z) = \frac{z + z^2/2}{1 + z + z^2/6}.$$

At $z = 1$, $\log 2 \approx 0.6931$:

Approximant	Coeffs used	Value at $z = 1$	Error
Taylor $z - z^2/2$ ($R_{2,0}$)	2	0.5000	-0.193
Padé $R_{1,1} = z/(1 + z/2)$	2	0.6667	-0.027
Padé $R_{2,2}$	4	0.6923	-0.0008

- Same Taylor information, the rational form is one to two orders of magnitude more accurate. Diagonal (p, p) Padé converges quadratically in the order, vs. linearly for Taylor: *more accuracy for the same data*.

Padé – Specialisation to e^z

- For $f(z) = e^z$, $c_k = 1/k!$, the Padé system has a closed form $R_{p,q}(z) = \left[\sum_{k=0}^{p} \binom{p}{k} \frac{(p+q-k)!}{(p+q)!} z^k\right] / \left[\sum_{k=0}^{q} \binom{q}{k} \frac{(p+q-k)!}{(p+q)!} (-z)^k\right]$.

- The first few entries of the **Padé table**:

	$q=0$	$q=1$	$q=2$
$p=0$	1	$\frac{1}{1-z}$	$\frac{1}{1-z+z^2/2}$
$p=1$	$1+z$	$\frac{1+z/2}{1-z/2}$	$\frac{1+z/3}{1-2z/3+z^2/6}$
$p=2$	$1+z+\frac{z^2}{2}$	$\frac{1+2z/3+z^2/6}{1-z/3}$	$\frac{1+z/2+z^2/12}{1-z/2+z^2/12}$

$\Rightarrow$ $R_{p,q}$ is A-stable iff $q \geq p$, L-stable iff $q > p$.

Padé – A catalogue of numerical schemes

- Each Padé entry of e^z *is* a one-step scheme for the test problem — and, after $z \mapsto M\,dt$, for any linear system:

Padé entry	Scheme	Global order / stability
$(1,0)$	Forward Euler	1, conditional
$(0,1)$	Backward Euler	1, A- & L-stable
$(1,1)$	**Crank–Nicolson** / impl. midpoint	2, A-stable
(s,s)	Gauss–Legendre (s stages)	$2s$, A-stable, symmetric
$(s{-}1,s)$	Radau IIA (s stages)	$2s{-}1$, L-stable
$(s{-}1,s{-}1)$	Lobatto IIIA (s stages)	$2s{-}2$, A-stable, endpoint nodes

- **Crank–Nicolson is the default**: the smallest A-stable, second-order entry — one implicit solve per step. The higher families are now implemented too (`scheme=gauss` / `radau` / `lobatto_iiia`, with `order=`); Gauss 2 is exactly Crank–Nicolson, and Lobatto IIIA 2 is the trapezoidal rule. Radau IIA is the L-stable choice for stiff problems.

Plan

Crank–Nicolson – The implicit midpoint formula

- The $(1,1)$-Padé approximant of e^z is $R_{1,1}(z) = (1 + z/2)/(1 - z/2)$. Applied to the test problem $\dot{x} = \lambda x$, $(1 - \lambda dt/2)\, x_{i+1} = (1 + \lambda dt/2)\, x_i$, re-arranged as a finite-difference equation:

$$\boxed{\frac{x_{i+1} - x_i}{dt} \;=\; \lambda\, \tfrac{1}{2}(x_i + x_{i+1}).}$$

- For a general right-hand side $\dot{x} = g(x,t)$ there are two equivalent realisations of the same $(1,1)$-Padé: the **trapezoidal rule** $\frac{x_{i+1}-x_i}{dt} = \frac{1}{2}\big[g(x_i,t_i) + g(x_{i+1},t_{i+1})\big]$ (the average of forward Euler $x_{i+1} = x_i + dt\, g(x_i,t_i)$ and backward Euler $x_{i+1} = x_i + dt\, g(x_{i+1},t_{i+1})$) and the **implicit midpoint rule** $\frac{x_{i+1}-x_i}{dt} = g\big(\frac{x_i+x_{i+1}}{2},\, t_{i+\frac{1}{2}}\big)$ (evaluation at the midpoint).

- The two coincide on linear problems and to leading order on smooth nonlinear ones. Continuo uses the implicit midpoint form: one residual evaluation per interval, and the natural realisation for implicit DAEs.

Crank–Nicolson – Properties (summary)

- **Second-order accurate.** Local truncation $O(dt^3)$, global error $O(dt^2)$. Halving dt divides the error by ~ 4.
- **A-stable.** $|A(z)| \leq 1$ on the whole left half-plane: stable for any $dt > 0$. The user picks dt for accuracy alone.
- **Symmetric in time.** $A(z)A(-z) = 1$ on linear problems: reversing time and re-integrating recovers the same iterate exactly. No time direction is favoured — fitting for a BVP pinned from both the past (states) and the future (jumps).
- **Implicit.** One linear (or, for non-linear models, Newton-solved) system per step. In a BVP solve, all steps are coupled anyway — no extra cost.
- **Not L-stable.** $|A(z)| \to 1$ as $|z| \to \infty$ along the negative real axis: very stiff modes are damped only slowly. For genuinely stiff problems Radau IIA is preferable.

Crank–Nicolson – For implicit DAEs

- Continuo's model is implicit: $F(\dot{x}, x, e, \theta, t) = 0$. The Crank–Nicolson realisation on $[t_i, t_{i+1}]$ takes:

$$\dot{x} \;\leftarrow\; \frac{x_{i+1} - x_i}{dt} \quad \text{(applied only to dynamic components)},$$

$$x \;\leftarrow\; \tfrac{1}{2}(x_i + x_{i+1}), \qquad e \;\leftarrow\; e(t_{i+\frac{1}{2}}), \qquad t \;\leftarrow\; t_{i+\frac{1}{2}}.$$

- The per-interval residual is therefore

$$R_i(x_i, x_{i+1}) \;=\; F\left(\frac{x^d_{i+1} - x^d_i}{dt},\; \tfrac{x_i + x_{i+1}}{2},\; e(t_{i+\frac{1}{2}}),\; \theta,\; t_{i+\frac{1}{2}}\right),$$

  where x^d collects the n_d dynamic components.

- Implemented in `src/continuo/solve/disc/crank_nicolson.py`; built as a CasADi `Function` once, then evaluated symbolically at every interval when assembling the stacked system.

Plan

Example – A scalar ODE we can solve by hand

- Consider the simplest linear stable ODE,

$$\dot{x} \;=\; -\lambda x, \qquad x(0) = x_0, \qquad \lambda > 0.$$

  Exact solution: $x(t) = x_0 e^{-\lambda t}$.

- This is the linearisation of any stable economic relation around a steady state (the eigenvalue $-\lambda$ sets the speed of convergence). Here it has *one* state, *no* jump, so we drop the terminal condition.

- **Plan.** Apply the three schemes — forward Euler, backward Euler, Crank–Nicolson — on a uniform grid with step $dt = T/N$, derive each amplification factor, expose the defects of the two Eulers, and only then compare numerically.

Example – Forward Euler and its defects

- Apply $x_{i+1} = x_i + dt \cdot g(x_i, t_i)$ with $g(x) = -\lambda x$:

$$\boxed{x_{i+1} \;=\; (1 - \lambda\, dt)\, x_i, \qquad A_{\mathrm{FE}}(z) = 1 + z, \quad z = -\lambda\, dt.}$$

- **Accuracy.** Compare to $e^{-\lambda dt} = 1 - \lambda dt + \frac{1}{2}(\lambda dt)^2 + O\big((\lambda dt)^3\big)$: the $(\lambda dt)^2$ term is missing entirely. Local error per step $\frac{1}{2}(\lambda dt)^2 x_i = O(dt^2)$, hence **global order 1**.

- **Stability.** $|A_{\mathrm{FE}}| < 1 \iff 0 < \lambda dt < 2$: *conditional* stability with the CFL-type bound $dt < 2/\lambda$. Beyond it the iterate diverges.

- **Qualitative failure modes.**
  - $\lambda dt = 1$: $A = 0$, the iterate collapses to $x = 0$ after one step (technically stable, but useless),
  - $1 < \lambda dt < 2$: $A < 0$, the iterate *oscillates* around zero — the true solution is monotonic,
  - $\lambda dt = 2$: $A = -1$, sustained oscillation at constant amplitude (marginally stable),
  - $\lambda dt > 2$: $A < -1$, divergence.

Example – Backward Euler and its defects

- Apply $x_{i+1} = x_i + dt \cdot g(x_{i+1}, t_{i+1})$, again with $g(x) = -\lambda x$. Solve the implicit equation $(1 + \lambda dt)\, x_{i+1} = x_i$:

$$x_{i+1} \;=\; \frac{1}{1+\lambda\, dt}\, x_i, \qquad A_{\mathrm{BE}}(z) = \frac{1}{1-z}, \quad z = -\lambda\, dt.$$

- **Accuracy.** Expand $\frac{1}{1+\lambda dt} = 1 - \lambda dt + (\lambda dt)^2 - \cdots$. Coefficient on $(\lambda dt)^2$ is 1, not $\frac{1}{2}$: local error $\frac{1}{2}(\lambda dt)^2 x_i = O(dt^2)$ (opposite sign to FE — BE *overestimates*), so again **global order 1**.

- **Stability.** $0 < A_{\mathrm{BE}} < 1$ for any $\lambda dt > 0$: A-stable, monotonic decay, no dt restriction. *No qualitative failure mode*, unlike FE.

- **Default.** As $\lambda dt \to \infty$, $A_{\mathrm{BE}} \to 0$ (BE is also L-stable). Fast modes are damped too aggressively — fine for genuinely stiff problems, but on smooth macro dynamics this kills information the user may want to see. And the price (implicit solve per step) buys only first-order accuracy.

Example – Crank–Nicolson on this problem

- Same model in implicit DAE form: $F(\dot{x}, x) = \dot{x} + \lambda x = 0$. One dynamic row, no algebraic row. Per-interval residual on $[t_i, t_{i+1}]$:

$$R_i(x_i, x_{i+1}) \;=\; \underbrace{\frac{x_{i+1} - x_i}{dt}}_{\text{discrete } \dot{x}} + \lambda\, \underbrace{\tfrac{1}{2}(x_i + x_{i+1})}_{\text{midpoint } x} \;=\; 0.$$

- Setting $R_i = 0$ and isolating x_{i+1}:

$$\boxed{x_{i+1} \;=\; \frac{1 - \frac{1}{2}\lambda\, dt}{1 + \frac{1}{2}\lambda\, dt}\, x_i, \qquad A_{\mathrm{CN}}(z) \;=\; \frac{1 + z/2}{1 - z/2}.}$$

- Taylor-expand: $A_{\mathrm{CN}}(-\lambda dt) = 1 - \lambda dt + \frac{1}{2}(\lambda dt)^2 - \frac{1}{4}(\lambda dt)^3 + \cdots$ — matches $e^{-\lambda dt}$ through the $(\lambda dt)^2$ term, so the local error per step is $\left(\frac{1}{4} - \frac{1}{6}\right)(\lambda dt)^3 = \frac{1}{12}(\lambda dt)^3$. **Second-order** global accuracy and A-stable for any $\lambda dt > 0$.

Example – The three amplification factors

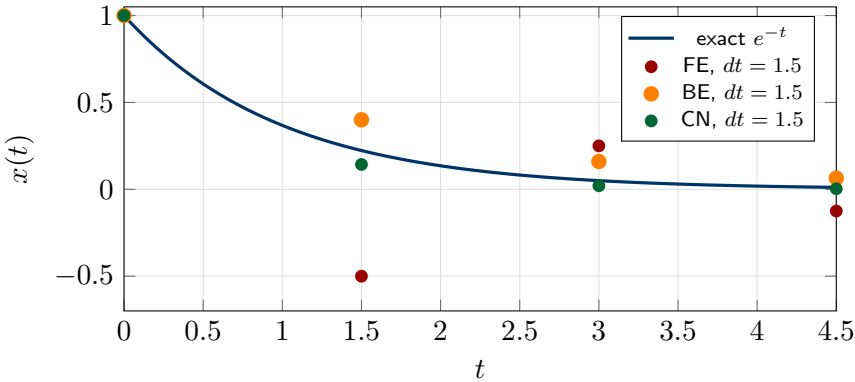
- Summary on the test problem $\dot{x} = -\lambda x$, $z = -\lambda dt$:

	$A(z)$	global order	stability
Forward Euler	$1 + z$	1	cond. $\lambda dt < 2$
Backward Euler	$1/(1 - z)$	1	A-stable, L-stable
Crank–Nicolson	$(1 + z/2)/(1 - z/2)$	2	A-stable

- All three are Padé approximants of e^z — $(1, 0)$, $(0, 1)$ and $(1, 1)$ respectively — so the table is a slice of the Padé table seen earlier. CN is the unique entry that is both A-stable *and* second-order: at fixed work per step (one linear solve), CN delivers an extra order of accuracy that BE leaves on the table.

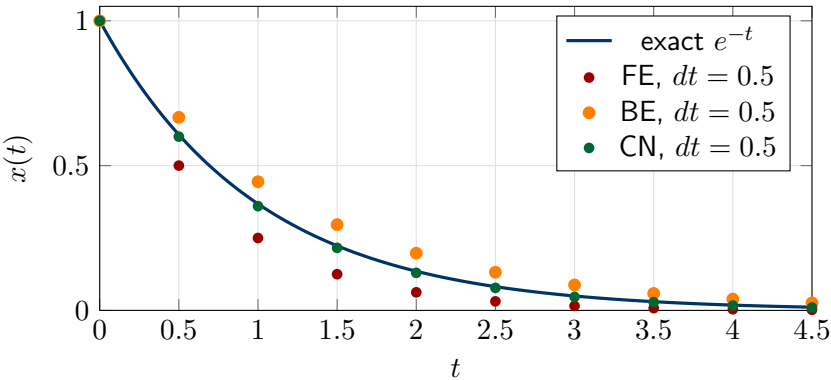
- Bottom line: FE pays for an unstable iteration with a fine grid; BE buys stability at the cost of an order of accuracy; CN pays the same implicit price as BE and gets the second order back.

Example – Numerical comparison (λdt moderately large)



$\lambda = 1$, $T = 4.5$, $x_0 = 1$, $N = 3$ ($\lambda dt = 1.5$). FE oscillates around zero (since $1 < \lambda dt < 2$ gives $A < 0$); BE overestimates the decay (always positive but biased upwards); CN already underestimates because λdt is large. All three fail visibly on a coarse grid, each in its own way.

Example – Numerical comparison (refined grid)



Same problem, refined to $dt = 0.5$ ($\lambda dt = 0.5$). FE now decays monotonically but stays too low; BE stays too high; CN is visually on top of the exact curve. Halving dt divides CN's error by four ($O(dt^2)$) and the Eulers' by two ($O(dt)$).

Nonlinear example – Solow growth and its closed form

- A genuinely *nonlinear* state equation that still has an exact solution: the Solow growth model

$$\dot{k} \;=\; s\,k^{\alpha} - \delta k, \qquad 0 < \alpha < 1,$$

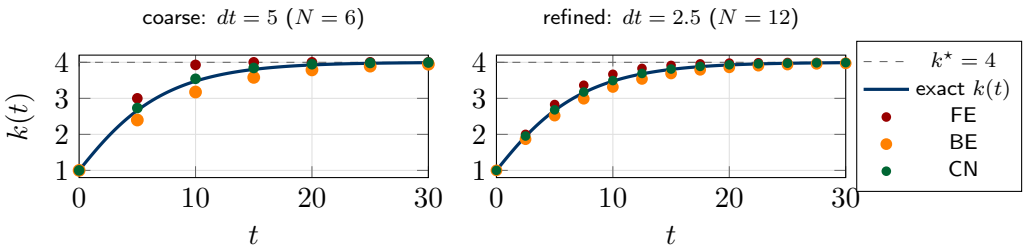
  s the saving rate, δ depreciation. One state (k), no jump — same boundary structure as the linear example, but the concave term k^{α} makes it nonlinear.
- **Bernoulli trick.** Substitute $z = k^{1-\alpha}$: $\dot{z} = (1-\alpha)k^{-\alpha}\dot{k} = (1-\alpha)\big(s - \delta z\big)$ — a *linear* ODE. Hence the exact path

$$k(t) = \left[\tfrac{s}{\delta} + \left(k_0^{1-\alpha} - \tfrac{s}{\delta}\right) e^{-(1-\alpha)\delta\, t}\right]^{\frac{1}{1-\alpha}}.$$

- **Steady state and speed.** $k^{\star} = (s/\delta)^{1/(1-\alpha)}$, approached at rate $(1-\alpha)\delta$ — exactly the local eigenvalue $g'(k^{\star}) = s\alpha(k^{\star})^{\alpha-1} - \delta = -(1-\alpha)\delta$. The Bernoulli map linearises the dynamics *globally*, so local and global convergence rates coincide.
- An exact yardstick on a nonlinear problem: only the scheme is approximated, never the truth.

Nonlinear example – Crank–Nicolson vs. the closed form

- Implicit-midpoint residual with $g(k) = s\,k^{\alpha} - \delta k$: $\dfrac{k_{i+1} - k_i}{dt} = g\left(\dfrac{k_i + k_{i+1}}{2}\right)$. g nonlinear, so each step is a scalar *nonlinear* equation in k_{i+1}, solved by Newton — a 1-D preview of the stacked solve.

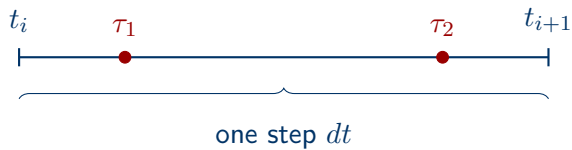


$\alpha = \frac{1}{2}$, $s = 0.8$, $\delta = 0.4$ ($k^\star = 4$), $k_0 = 1$, $T = 30$. Coarse grid magnifies the split (FE over, BE under, CN closest); halving dt divides CN's max error by four ($0.068 \to 0.018$) — **second order survives**.

Collocation – Sampling the dynamics

- To step from x_i to x_{i+1} we need the slope $\dot{x} = g(x, t)$ *inside* the interval. Crank–Nicolson uses **one** sample, at the midpoint.

- For more accuracy, sample the slope at s interior **stage points** $\tau_j = t_i + c_j\, dt$ (with $0 \le c_j \le 1$) and combine them — a **Runge–Kutta** method:



- *Collocation* picks the c_j as good quadrature nodes; the $s = 1$ midpoint choice *is* Crank–Nicolson.

Collocation – Reading a Butcher tableau

- A Runge–Kutta method is fully described by a small table — the **Butcher tableau**:

$$\begin{array}{c|c} c & A \\ \hline & b^\top \end{array}$$

c_j : *where* stage j sits in the step;
b_j : its *weight* in the final update;
$A_{j\ell}$: how stage j uses the other stages.

- The one-stage methods read at a glance (update formula recalled below each):

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

forward Euler

$x_{i+1} = x_i + dt\, g(x_i, t_i)$

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

backward Euler

$x_{i+1} = x_i + dt\, g(x_{i+1}, t_{i+1})$

$$\begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array}$$

impl. midpoint = CN

$x_{i+1} = x_i + dt\, g(\frac{x_i + x_{i+1}}{2}, t_{i+\frac{1}{2}})$

one slope, taken at the start / end / middle of the step, full weight $b = 1$. A non-zero A means the stage is **implicit** (it depends on the unknown x_{i+1}).

Collocation – Stages and the "dynamic derivative"

- At each stage we need the slope. Only the **dynamic** variables (states + jumps) *have* a time derivative; the **algebraic** ones do not. So the unknown per stage is V_j, the derivative $\dot{x}$ of the dynamic components at τ_j — the *dynamic derivative*.

- The stage **state** is rebuilt from those slopes,

$$Z_j^d = x_i^d + dt \sum_\ell A_{j\ell} V_\ell,$$

(the algebraic part of Z_j comes out of the model). Two conditions per interval:

$$\underbrace{F\big(V_j, Z_j, e(\tau_j), \theta, \tau_j\big) = 0}_{\text{model holds at each stage}} \qquad \underbrace{x_{i+1}^d = x_i^d + dt \sum_j b_j V_j}_{\text{combine the slopes}}.$$

- **Check** $s = 1$ ($c = \frac{1}{2}, A = \frac{1}{2}, b = 1$): $V_1 = \frac{x_{i+1}^d - x_i^d}{dt}$ and $Z_1 = \frac{x_i + x_{i+1}}{2}$ — exactly Crank–Nicolson. CN is the one-stage member.

Collocation – Gauss–Legendre

- Interior nodes (the Gauss points), none at the ends; **symmetric**, **A-stable**, order $2s$. $s{=}1$ is Crank–Nicolson; the two-stage member (order 4), with stage slopes k_j and $c_{1,2} = \frac{1}{2} \mp \frac{\sqrt{3}}{6}$, is

$$\begin{array}{c|cc} \frac{1}{2}-\frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4}-\frac{\sqrt{3}}{6} \\ \frac{1}{2}+\frac{\sqrt{3}}{6} & \frac{1}{4}+\frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

$$\begin{aligned} k_1 &= g\big(x_i + dt(\tfrac{1}{4}k_1 + (\tfrac{1}{4}-\tfrac{\sqrt{3}}{6})k_2),\ t_i + c_1 dt\big), \\ k_2 &= g\big(x_i + dt((\tfrac{1}{4}+\tfrac{\sqrt{3}}{6})k_1 + \tfrac{1}{4}k_2),\ t_i + c_2 dt\big), \\ x_{i+1} &= x_i + \tfrac{dt}{2}\,(k_1 + k_2). \end{aligned}$$

- Stages → order: $s = 1, 2, 3 \Rightarrow 2, 4, 6$. `scheme=gauss, order=4`.

Collocation – Radau IIA

- Last node at the **right endpoint** ($c_s = 1$); **stiffly accurate** (b is the last row of A), **L-stable**, order $2s-1$. $s{=}1$ is backward Euler; the two-stage member (order 3) is

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}$$

$$\begin{aligned} k_1 &= g\big(x_i + dt(\tfrac{5}{12}k_1 - \tfrac{1}{12}k_2),\ t_i + \tfrac{dt}{3}\big), \\ k_2 &= g\big(x_i + dt(\tfrac{3}{4}k_1 + \tfrac{1}{4}k_2),\ t_{i+1}\big), \\ x_{i+1} &= x_i + dt(\tfrac{3}{4}k_1 + \tfrac{1}{4}k_2) \ \ (= \text{stage 2}). \end{aligned}$$

- Stages → order: $s = 1, 2, 3 \Rightarrow 1, 3, 5$. `scheme=radau, order=5`.

Collocation – Lobatto IIIA

- **Both endpoints** are nodes ($c_1 = 0$, $c_s = 1$), so the first row of A vanishes ($k_1 = g(x_i, t_i)$ is explicit). **A-stable**, order $2s-2$ — the classic BVP collocation (`bvp4c`). s=2 is the trapezoidal rule; the three-stage member (order 4) is

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{5}{24} & \frac{1}{3} & -\frac{1}{24} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

$$\begin{aligned} k_1 &= g(x_i,\, t_i), \\ k_2 &= g\big(x_i + dt(\tfrac{5}{24}k_1 + \tfrac{1}{3}k_2 - \tfrac{1}{24}k_3),\; t_{i+\frac{1}{2}}\big), \\ k_3 &= g\big(x_i + dt(\tfrac{1}{6}k_1 + \tfrac{2}{3}k_2 + \tfrac{1}{6}k_3),\; t_{i+1}\big), \\ x_{i+1} &= x_i + dt(\tfrac{1}{6}k_1 + \tfrac{2}{3}k_2 + \tfrac{1}{6}k_3). \end{aligned}$$

- Stages → order: $s = 2, 3, 4 \Rightarrow 2, 4, 6$. `scheme=lobatto_iiia, order=4`.

Plan

Stacked system – Dynamic vs. algebraic rows

- Of the n rows of F, only the n_d **dynamic** rows involve $\dot{x}$. Crank–Nicolson collocates these on each interval (N blocks). (The dynamic *variables* fill the first n_d columns – states then jumps – so $\dot{x}$ hits a contiguous block; the rows themselves may sit in any order.)

- The n_a **algebraic** rows have $\dot{x}$ absent: there is nothing to discretise. They are enforced **pointwise** at every grid point ($N+1$ blocks).

- Equation count, matching the $n(N+1)$ unknowns exactly:

Source	Per slot	Count
Dynamic rows (CN collocation)	n_d	$N \cdot n_d$
Algebraic rows (pointwise)	n_a	$(N+1) \cdot n_a$
Initial states	1	n_S
Terminal jumps	1	n_J
Total		$n(N+1)$

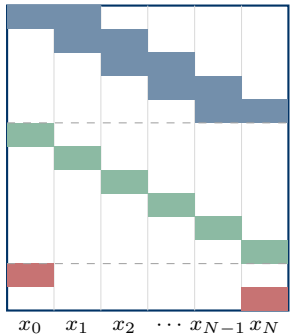
Stacked system – Assembly

- Stack the unknowns $X = (x_0, \ldots, x_N) \in \mathbb{R}^{n(N+1)}$ and assemble the residual $G(X)$ by stacking four row-groups, *in this order*:

  1. dynamic rows of $R_i(x_i, x_{i+1})$, for $i = 0, \ldots, N-1$ (N blocks of n_d);
  2. algebraic rows of F at every grid point ($N{+}1$ blocks of n_a);
  3. initial-condition equations $x_{0,s} - x_s^0$ for each state (n_S scalars);
  4. terminal-condition equations $x_{N,j} - \bar{x}_j$ for each jump (n_J scalars).

- Total: $n(N{+}1)$ equations in $n(N{+}1)$ unknowns. The assembly is symbolic (CasADi), so the Jacobian comes for free with `ca.jacobian(g, X)` — exact, sparse, and machine-precision.

Stacked system – Sparsity pattern of $J = \partial G / \partial X$



dynamic rows (CN)
N blocks, upper-bidiagonal
(block i couples x_i, x_{i+1})

algebraic rows
$N{+}1$ blocks, diagonal

boundary rows
pin x_0 (states), x_N (jumps)

Grouping the rows as on the previous slide — the N dynamic CN blocks first, then the $N{+}1$ algebraic blocks, then the two boundary rows — J is **not** a single banded matrix: an upper-bidiagonal dynamic stripe (block-row i touches x_i, x_{i+1}) on top of a diagonal algebraic stripe, closed by boundary entries pinning x_0, x_N. Non-zeros $O(n^2 N)$. The pattern is **constant** across Newton steps, so the symbolic factorisation is done once and only refreshed (next slides), keeping each solve cheap and linear in N.

Stacked system – Newton

- At iterate $X^{(k)}$, solve the sparse linear system

$$J\big(X^{(k)}\big)\,\Delta X \;=\; -\,G\big(X^{(k)}\big),$$

  then update $X^{(k+1)} = X^{(k)} + \alpha\,\Delta X$ with a step $\alpha \in (0, 1]$.

- In Continuo: a **pluggable** sparse solver (default `klu`+BTF, SuperLU fallback — see next slides), a CasADi-supplied exact sparse Jacobian, $\|G\|_\infty$ convergence criterion (10^{-10}), max 50 iterations.

- A simple **backtracking line search** halves α until $\|G(X^{(k+1)})\|_\infty < \|G(X^{(k)})\|_\infty$, which prevents pure-Newton overshoots far from the solution.

- **Initial guess.** Default: the terminal steady state replicated at every grid point. For mildly non-linear models this is enough; for harder problems one can warm-start from a coarser grid or from a homotopy.

Stacked system – The linear core

- Every Newton step solves one sparse system $J\,\Delta X = -G$. The **sparsity pattern of J is constant** — across Newton iterations, and (at a fixed grid) across the segments of a multi-segment run.

- So the **symbolic** work (fill-reducing ordering, block structure) is done *once* and only the **numeric** factorisation is refreshed. Continuo exposes this as a pluggable interface,

  $$\texttt{analyze} \rightarrow \texttt{factor} \rightarrow \texttt{refactor} \rightarrow \texttt{solve},$$

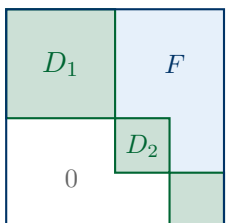
  with `analyze` hoisted out of the loop, so each step is just `refactor + solve`.

- Backends: `superlu` (SciPy, always available — the fallback), `klu` (SuiteSparse, the **default**), `umfpack`, `pardiso`; selected by `auto` unless pinned.

Stacked system – KLU + BTF, step by step

- **Sparse LU.** A direct solver factors $J = LU$ (L lower, U upper triangular); the solve is then two triangular sweeps. Storing L, U takes *more* nonzeros than J — the surplus is **fill-in**, and the row/column *ordering* is chosen to keep it small.

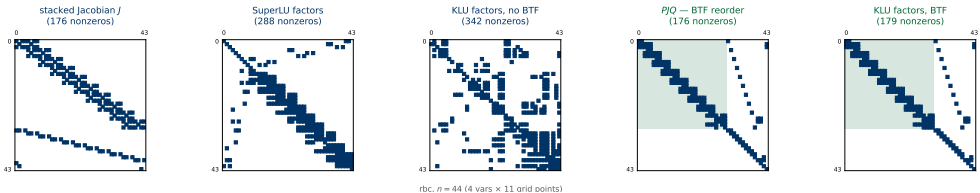
- **BTF.** KLU first permutes rows and columns so that PJQ is **block upper-triangular**, then factorises only the diagonal blocks:



diagonal blocks $D_k = L_k U_k$ — factored
off-diagonal F — kept, applied by back-substitution
below the blocks — zero

So the LU **fill-in stays inside** the diagonal blocks; the off-diagonal F is kept untouched. KLU stores $\{L_k, U_k\}$ (factored, with fill) + F (kept) — the next slide shows this on a real model.

Stacked system – Factorisations of J



- **BTF** (`klu`) reorders J into PJQ (block upper-triangular, 4th panel). Because J is **block-bidiagonal** (CN's two-point stencil), this order is almost *fill-free*: the fill is a small **constant** ($176 \to 179$), independent of the horizon. SuperLU's fill, in contrast, **grows** with N ($176 \to 288$); without BTF KLU fills even more (342).

- BTF only peels the algebraic + boundary rows as 1×1 blocks — the dynamic core stays one big block. Less fill is cheaper, but the *bigger* per-iteration win is **reusing** the factorisation (next slide).

Stacked system – Why KLU here

KLU is about $7\times$ **faster** per Newton step than SuperLU — *not* mainly because of BTF, but three effects:

effect	gain	why
amortised symbolic	$\sim 3\times$	analyse once; each step is a cheap `refactor` (SuperLU re-factorises in full)
KLU's sparse LU	$\sim 1.6\times$	engine tuned for near-banded matrices
BTF pre-ordering	$\sim 1.2\times$	peels the algebraic singletons

- **Reuse is the real win**: the constant pattern is analysed once, shared across Newton steps *and* segments.
- **Reuse alone is not enough**: UMFPACK *also* reuses the analysis, yet its slow numeric only *ties* SuperLU — KLU adds a fast numeric.
- A **multi-step** scheme (BDF — a three-point stencil, as in Dynare) makes J *block-tridiagonal*: BTF no longer helps, and a **banded** solver (block-Thomas) fits instead. So the linear core is **pluggable**, `auto`-routed by the scheme's stencil.

Stacked system – Anticipated vs. surprise shocks

- Within a single **information segment**, all future exogenous values are known: $e(t)$ is just a deterministic function evaluated at the midpoints.
- A **surprise** corresponds to a change in the information set at some reveal time t_r. Continuo's orchestrator partitions $[0, T]$ into segments separated by reveal times and solves *one* perfect-foresight problem per segment:
  - the segment's grid extends a full horizon T past t_r (agents believe the new beliefs hold forever),
  - the state at t_r is carried from the previous segment (states cannot jump),
  - the jump variables re-optimise at t_r given the new beliefs.
- Each segment uses the *same* CN+Newton machinery — surprises are just a sequence of perfect-foresight solves, glued together.

Plan

Grids – Shock-aligned nodes

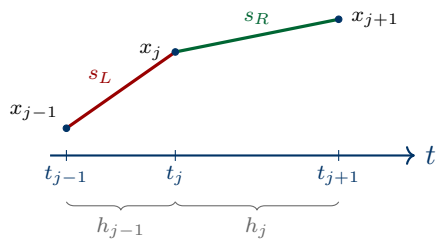
- A reveal time t_r is a **kink** in the path. Continuo builds each segment's grid with an *exact node* at t_r, so the kink is resolved cleanly instead of being smeared across the nearest interval.
- This is automatic and needs no tuning: the segment boundaries of the anticipated/surprise machinery are placed on the mesh by construction.
- It is the cheap, sure win — but it only handles the kinks we *know* in advance. An occasionally-binding constraint puts a kink where the solution decides, not where a shock is revealed.

Grids – Adaptive refinement: equidistribution

- A uniform grid wastes nodes on the easy stretches and starves the hard ones. Instead attach a **difficulty weight** w_i to each interval and make it *equal* across intervals — the **equidistribution** principle (de Boor; COLSYS; `bvp4c`/`bvp5c`).
- The loop: *solve* on the current mesh, *measure* w_i, *bisect* the intervals whose w_i exceeds the average, *re-solve* — until a global error estimate drops below the tolerance, or a node cap is reached.
- Refinement only *adds* nodes, so reveal and terminal times stay pinned. Switched on with `simulate(..., adapt=1e-5, monitor=...)`.

Grids – The curvature monitor (where to refine)

- The interpolation error on an interval scales like $h_i^2\,|x''|$, so the hard intervals are where the solution **bends**. Curvature is the **change of slope** between the two neighbouring secants:



$$s_L = \tfrac{x_j - x_{j-1}}{h_{j-1}}, \quad s_R = \tfrac{x_{j+1} - x_j}{h_j}, \qquad |x_j''| \approx \frac{|\,s_R - s_L\,|}{\frac{1}{2}(h_{j-1} + h_j)} \quad \text{(the analogue of } \tfrac{x_{j-1} - 2x_j + x_{j+1}}{h^2}\text{)}.$$

Each variable is scaled by its range; the interval weight $w_i = h_i\,|x''| \approx \int_{I_i} |x''|$ is equidistributed — $\max$/mean of w_i is the `equidistribution_ratio`.

Grids – When to stop (1): Richardson

- We need a single **magnitude** for the tolerance test. Solve the *whole* path twice — on the mesh (N) and on the **bisected** mesh ($2N$, every interval halved, so the $N+1$ nodes survive) — and compare at those shared nodes. With order p and exact path $x^\star$:

$$x_N - x^\star \approx C\,h^p, \quad x_{2N} - x^\star \approx 2^{-p} C\,h^p \implies \boxed{e_N \approx \tfrac{2^p}{2^p-1}\,\|x_N - x_{2N}\|},$$

  with $\|\cdot\|$ the **max over nodes and variables**; C and $x^\star$ cancel.

- This sizes the **global** error only — the doubled mesh is a throwaway *measurement*, not the refinement. *Where* to add nodes stays the curvature monitor's call (previous slide); Richardson just says *whether* to stop. Cost: one full extra solve per pass.

- **Caveat.** It assumes the *nominal* order p, so it badly under-estimates the error at a **kink** and stops too early (next slide).

Grids – Why Richardson under-refines at a kink

- The factor $\frac{2^p}{2^p-1}$ bakes in one assumption: each bisection divides the error by 2^p, so the $2N$ solution is far more accurate and the gap $\|x_N - x_{2N}\|$ is essentially the whole coarse error.

- **Smooth** (Gauss 4, $p = 4$): factor $= \dfrac{2^4}{2^4 - 1} = \dfrac{16}{15} \approx 1.07$ — the gap *is* the error. All good.

- **Kink** — a *corner* (continuous but the slope jumps: $\mathscr{C}^0$, not $\mathscr{C}^1$): refining cuts the error by only ≈ 2 (order 1), not 2^p. The honest factor is $\dfrac{2^1}{2^1 - 1} = 2$, so the gap is just *half* the true error.

- Richardson still multiplies by 1.07, so it reports about *half* the real error — meets the tolerance prematurely and stops while the kink is under-resolved. `residual` assumes no order, so it does not fall for this.

Grids – When to stop (2): the residual defect

- **Residual / defect.** Fit a smooth interpolant $\tilde{x}(t)$ through the node values (a cubic spline), put it back into the ODE, and measure how far it misses, at the interval midpoints:

$$r_i \;=\; \left\| \tilde{x}'(t_{i+\frac{1}{2}}) - g\big(\tilde{x}(t_{i+\frac{1}{2}}),\, t_{i+\frac{1}{2}}\big) \right\|.$$

- **No order assumption** and **no extra solve**, so it stays reliable at a **kink** — it keeps refining where Richardson would quit. (`bvp5c` uses a calibrated version of this defect for Lobatto IIIA.)

- **Default**: curvature for placement + `residual` for the stop — cheap, and robust at the kinks adaptivity is meant for. Switch to `monitor=richardson` for a calibrated tolerance on a smooth solve.

Experiments – A zero-lower-bound test bed

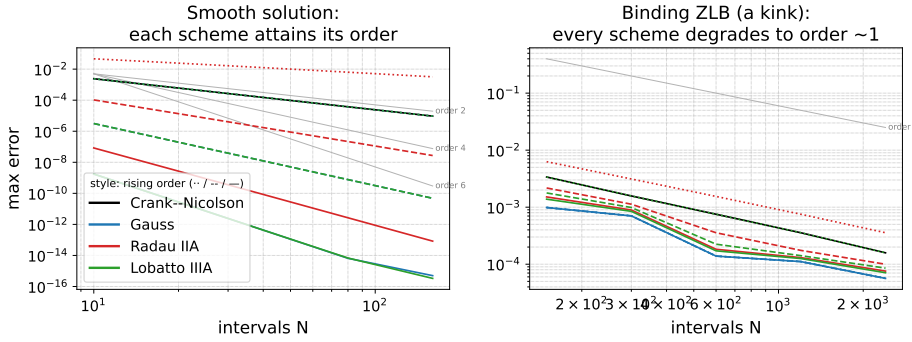
- Nonlinear New-Keynesian model (`examples/nk-nonlinear`): forward-looking consumption C and inflation π, with the policy rate at the **ZLB**,

$$R \;=\; \max\bigl(0,\; \rho + \phi_\pi\, \pi\bigr).$$

- An anticipated 12% TFP boom over $[0, 3)$ lowers marginal cost $\Rightarrow$ deflation $\Rightarrow$ the Taylor rule wants a negative rate. The ZLB **binds on** $\approx [0, 2.46]$ (π trough -6.6%).

- The $\max$ makes the solution a *corner* — continuous, but with a jump in a derivative (only C^0/C^1, no smoother) — at the binding/exit times; and the **exit time is endogenous** (where π crosses $-\rho/\phi_\pi$), so it does *not* land on a node a priori. This is the hard case for the schemes.

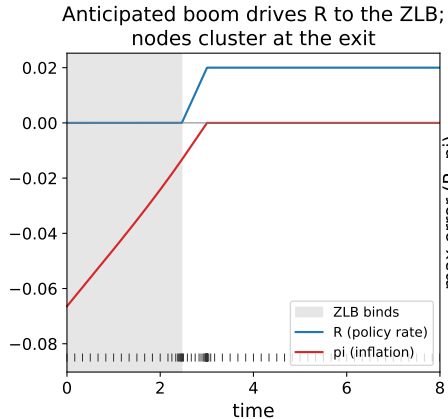
Experiments – Order: smooth vs. the kink

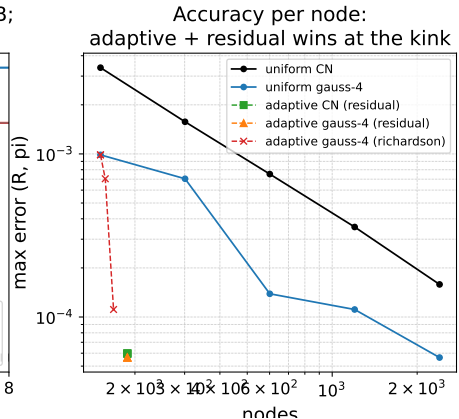


Smooth (left): each family attains its order. Binding ZLB (right): all collapse to order ≈ 1 — the kink, not the scheme, sets the accuracy.

Why order 1? The ZLB exit time is endogenous and rarely a node; a fixed mesh locates it only to within $O(h)$, and a *corner displaced by $O(h)$ gives an $O(h)$ error* — order 1, for *any* scheme. The only cure is to put nodes there — i.e. refine adaptively.

Experiments – Adaptive refinement at the kink





Adaptive nodes cluster at the endogenous ZLB exit (left). Per node, adaptive wins: adaptive CN reaches 6×10^{-5} with **188 nodes** — the accuracy uniform Gauss 4 needs **2401** nodes for (right).

Experiments – The error monitor matters

adaptive solve (tol 10^{-5})	nodes	max error
Crank–Nicolson, `residual`	188	6.0×10^{-5}
Gauss 4, `residual`	188	5.6×10^{-5}
Gauss 4, `richardson`	157	7.0×10^{-4}

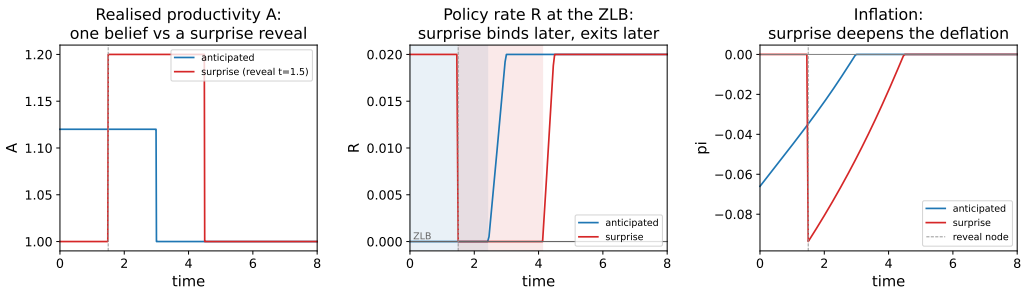
- `richardson` rescales the coarse/fine gap by the *nominal* order. At a kink the realised order is ≈ 1, so it **underestimates** the error and stops refining too early.

- `residual` measures the actual ODE defect of the solution — no order assumption — so it keeps refining at the kink and converges.

Rule of thumb

At a kink, refine with the `residual` monitor; reserve `richardson` for smooth solves where its calibration holds.

Experiments – Anticipated vs. surprise



Anticipated: one belief, one segment. **Surprise:** a reveal at $t = 1.5$ splits the horizon into two segments glued at an exact node; the ZLB-exit kink inside each segment stays endogenous (adaptive still helps).

Experiments – What to use when

- **Smooth solution:** a higher-order scheme on a modest uniform grid — Gauss for accuracy, Radau IIA when stiff.
- **Known kink** (a reveal): nothing to do — it is placed on a node automatically.
- **Unknown kink** (occasionally-binding constraint): adaptive refinement with the `residual` monitor; the scheme order barely matters there, so Crank–Nicolson is a fine, robust base.
- *Schemes win when the solution is smooth; grids win at kinks.*

Plan

Comparison – Side by side

	Dynare	**Continuo (Crank–Nicolson)**
Model	$f(y_{t+1}, y_t, y_{t-1}, u_t, \theta) = 0$	$F(\dot{x}, x, e, \theta, t) = 0$
Time	intrinsically discrete	continuous, discretised on a grid
Step size	1 period	free $dt = T/N$
Discr. error	none	$O(dt^2)$ globally
Stencil per eqn.	3 points $(t{-}1, t, t{+}1)$	2 points (t_i, t_{i+1})
Jacobian	block tridiagonal	block bidiagonal + block diagonal
Default solver	sparse direct (UMFPACK)	KLU + BTF (auto), SuperLU fallback
Boundary data	y_0, $\bar{y}_{T+1}$ for all	$x_S(0)$, $\bar{x}_J(T)$ split by type
Algebraic vars	like any equation	detected, enforced pointwise
Symbolic engine	custom C preprocessor	CasADi (Python)

Comparison – Conceptual mapping

- Both methods reduce the BVP to a square non-linear system solved by Newton on a stacked vector of unknowns. The skeleton is identical:

$$G(X) = 0, \qquad X^{(k+1)} = X^{(k)} - \alpha\, J^{-1} G(X^{(k)}).$$

- **Why does Dynare have a tridiagonal Jacobian but Continuo only a bidiagonal one?** Because Crank–Nicolson couples *two* adjacent points; a discrete-time model can also reference y_{t-1}, coupling *three*.

- **Could one use a 3-point scheme in continuous time?** Yes (BDF2, trapezoidal-BDF blends). It would buy higher accuracy or L-stability but reintroduce a tridiagonal pattern.

- Conversely, one can read Crank–Nicolson as a discrete-time Dynare model by treating $\frac{1}{2}(x_i + x_{i+1})$ and $(x_{i+1} - x_i)/dt$ as new variables, the algebra is the same, the bookkeeping is just different.

Comparison – When does the difference matter?

- **Step-size flexibility.** In Dynare $dt = 1$ period is built in; refining requires reformulating the model at higher frequency. In Continuo dt is a numerical knob: choose N until $O(dt^2)$ accuracy is sufficient.

- **Calibration at the right frequency.** Continuous-time preferences and technologies are calibrated directly in flow units (rates per unit time). Discrete-time calibration must pick a convention (annual? quarterly?) and may suffer from time-aggregation bias for fast-moving variables.

- **Occasionally-binding constraints.** A constraint that binds for a short duration is naturally captured with a fine grid in continuous time; in discrete time the period length sets a lower bound on the resolution.

- **Symbolic differentiation.** Both pipelines need exact Jacobians. Continuo uses CasADi (operator overloading, automatic differentiation); Dynare uses a custom preprocessor that emits derivative code. Both are exact.

Plan

Conclusion – Take-aways

- Solving a perfect-foresight model = solving a two-point BVP = stacking the unknowns at every time slot and running Newton on a square block-sparse non-linear system.

- **Crank–Nicolson** is the default — second-order, A-stable, two-point stencil (the implicit-midpoint form). Beyond it, **higher-order collocation** (Gauss / Radau IIA / Lobatto IIIA) pays off on smooth solutions, while **adaptive grids** cluster nodes at kinks. The ZLB is the cautionary tale: at a kink the order collapses, and the lever is the grid (with the `residual` monitor), not the scheme.

- **Dynare** solves an analogous stacked Newton system but with an intrinsically discrete model, a 3-point stencil, and a sparse solve over its block-tridiagonal Jacobian.

- Both share one backbone — *symbolic Jacobian + sparse linear algebra + Newton* — differing in the pre-processing (discretise an ODE *vs.* accept a difference equation) and how the block structure is exploited.

Conclusion – Further reading

- **Crank–Nicolson, BVPs and collocation.** Ascher & Petzold (1998), *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Hairer & Wanner (1996), *Solving Ordinary Differential Equations II*. Iserles (2009), *A First Course in the Numerical Analysis of Differential Equations*.

- **Adaptive collocation and meshes.** de Boor (1973); Ascher, Mattheij & Russell (1995), *Numerical Solution of Boundary Value Problems for ODEs*; Kierzenka & Shampine (2001), the `bvp4c` residual-control solver. **Continuous-time perfect foresight:** Trimborn, Koch & Steger (2008), *Multidimensional transitional dynamics*, *Macroeconomic Dynamics*.

- **Dynare's algorithm.** Juillard (1996), *Dynare: A program for the resolution and simulation of dynamic models with forward variables through the use of a relaxation algorithm*, CEPREMAP. Adjemian *et al.* (2026), *Dynare Reference Manual*, Chap. "Perfect-foresight simulation".