Lecture 8

Continuous time dynamic models

Ivan Rudik
AEM 7130

Roadmap

1. The theory behind continuous time models
2. Numerical methods for solving continuous time model

Model setup

Consider a problem where each period an agent obtains flow utility $J(x(t), u(t))$, where x is our **state** and u is our **control**

Model setup

Consider a problem where each period an agent obtains flow utility $J(x(t), u(t))$, where x is our **state** and u is our **control**

Suppose there is a finite horizon with a **terminal time** T

Model setup

The agent's objective is to maximize the total payoff, subject to the transitions of the states

$$\max_{u, x_T} \int_0^T J(x(t), u(t))\, dt$$
$$\text{subject to: } \dot{x}(t) = g(x(t), u(t)),\ x(0) = x_0,\ x(T) = x_T$$

This is an open-loop solution so we optimize our entire policy trajectory from time $t = 0$

We will not be solving for functions of states, but functions of time: $u(t), x(t)$

Hamiltonians

In a dynamic optimization problem, we will have an auxiliary function that yields the first-order conditions

Hamiltonians

In a dynamic optimization problem, we will have an auxiliary function that yields the first-order conditions

This function is called the *Hamiltonian*:

$$H(x(t), u(t), \lambda(t)) \equiv J(x(t), u(t)) + \lambda(t) g(x(t), u(t))$$

It is a function that treats the transitions as quasi-constraints so it appears similar to the Lagrangian you know

Hamiltonians

Pontryagin's Maximum Principle states that the following conditions are necessary for an optimal solution:

$$
\begin{aligned}
\frac{\partial H(x(t), u(t), \lambda(t))}{\partial u} &= 0 \ \forall t \in [0, T] && \text{(Maximality)} \\
\frac{\partial H(x(t), u(t), \lambda(t))}{\partial x} &= -\dot{\lambda}(t) && \text{(Co-state)} \\
\frac{\partial H(x(t), u(t), \lambda(t))}{\partial \lambda} &= \dot{x}(t) && \text{(State transitions)} \\
x(0) &= x_0 && \text{(Initial condition)} \\
\lambda(T) &= 0 && \text{(Transversality)}
\end{aligned}
$$

What do these conditions mean?

Necessary conditions

First, what *is* the Hamiltonian?

Necessary conditions

First, what *is* the Hamiltonian?

The Hamiltonian tells us the contribution of that instant t to overall utility via the change in flow utility and the change in the state (which affects future flow utilities)

Necessary conditions

First, what *is* the Hamiltonian?

The Hamiltonian tells us the contribution of that instant t to overall utility via the change in flow utility and the change in the state (which affects future flow utilities)

The decisionmaker can use her control to increase the contemporaneous flow utility and reap immediate rewards, or to alter the state variable to increase future rewards

$$H(x(t),u(t),\lambda(t)) \equiv \underbrace{J(x(t),u(t))}_{\text{current flow}} + \underbrace{\lambda(t)g(x(t),u(t))}_{\text{change in future value}}$$

Necessary conditions

$$\frac{\partial H(x(t), u(t), \lambda(t))}{\partial u} = 0 \ \forall t \in [0, T]$$

The **maximality condition**: in every instant, we select the control so that we can no longer increase our total payoff

Necessary conditions

$$\frac{\partial H(x(t), u(t), \lambda(t))}{\partial u} = 0 \ \forall t \in [0, T]$$

The **maximality condition**: in every instant, we select the control so that we can no longer increase our total payoff

It effectively sets the net marginal benefits of the control to zero

Necessary conditions

$$\frac{\partial H(x(t), u(t), \lambda(t))}{\partial \lambda} = \dot{x}(t)$$

The **state transition** is just a definition

Necessary conditions

$$\frac{\partial H(x(t), u(t), \lambda(t))}{\partial \lambda} = \dot{x}(t)$$

The **state transition** is just a definition

Taking the derivative of the Hamiltonian with respect to the shadow value, just like a Lagrangian, yields this constraint back

Necessary conditions

$$\frac{\partial H(x(t), u(t), \lambda(t))}{\partial x} = -\dot{\lambda}(t)$$

The **co-state condition** defines how the shadow value of our state transition, called the **co-state variable**, evolves over time

Necessary conditions

$$\frac{\partial H(x(t), u(t), \lambda(t))}{\partial x} = -\dot{\lambda}(t)$$

The co-state condition defines how the shadow value of our state transition, called the co-state variable, evolves over time

What is the co-state?

Necessary conditions

What is the co-state?

Necessary conditions

What is the co-state?

The additional future value of having one more unit of our state variable

Necessary conditions

What is the co-state?

The additional future value of having one more unit of our state variable

Suppose we increase today's stock of x by one unit and this increases the instantaneous change in our value (H)

Necessary conditions

What is the co-state?

The additional future value of having one more unit of our state variable

Suppose we increase today's stock of x by one unit and this increases the instantaneous change in our value (H)

Then the shadow value of that stock (λ) must decrease along an optimal trajectory

Necessary conditions

What is the co-state?

The additional future value of having one more unit of our state variable

Suppose we increase today's stock of x by one unit and this increases the instantaneous change in our value (H)

Then the shadow value of that stock (λ) must decrease along an optimal trajectory

Why?

Necessary conditions

If it didn't, we could increase value by accumulating more of the stock variable
$\rightarrow$ there is a profitable deviation and what we were doing cannot be optimal

We can re-write the co-state equation as

$$\frac{\partial J}{\partial x} + \lambda(t)\frac{\partial g}{\partial x} + \dot{\lambda}(t) = 0$$

Necessary conditions

$$\frac{\partial J}{\partial x} + \lambda(t)\frac{\partial g}{\partial x} + \dot{\lambda}(t) = 0$$

We must have that the a unit of the stock's value must change (third term), so that is exactly offsets the change in value from increasing the stock in the immediate instant of time

Necessary conditions

$$\frac{\partial J}{\partial x} + \lambda(t)\frac{\partial g}{\partial x} + \dot{\lambda}(t) = 0$$

We must have that the a unit of the stock's value must change (third term), so that is exactly offsets the change in value from increasing the stock in the immediate instant of time

The immediate value is made up of the actual utility payoff (first term), and the future utility payoff payoff from how increasing the stock today affects the stock in the future (second term)

Necessary conditions

These necessary conditions give us the shape of the optimal path but they do not tell us what the optimal path actually is

Necessary conditions

These necessary conditions give us the shape of the optimal path but they do not tell us what the optimal path actually is

Many different paths are consistent with these differential equations, depends on the constant of integration

Necessary conditions

These necessary conditions give us the shape of the optimal path but they do not tell us what the optimal path actually is

Many different paths are consistent with these differential equations, depends on the constant of integration

We need additional optimality conditions to use as constraints to impose the optimal path

Pinning down the optimal path

We have effectively two ODEs, one for $\dot{\lambda}(t)$ and one for $\dot{x}(t)$

Pinning down the optimal path

We have effectively two ODEs, one for $\dot{\lambda}(t)$ and one for $\dot{x}(t)$

We need two constraints

Pinning down the optimal path

We have effectively two ODEs, one for $\dot{\lambda}(t)$ and one for $\dot{x}(t)$

We need two constraints

Constraint 1: Initial condition on the state

Pinning down the optimal path

We have effectively two ODEs, one for $\dot{\lambda}(t)$ and one for $\dot{x}(t)$

We need two constraints

Constraint 1: Initial condition on the state

This directly pins down where the state path starts

Pinning down the optimal path

We have effectively two ODEs, one for $\dot{\lambda}(t)$ and one for $\dot{x}(t)$

We need two constraints

Constraint 1: Initial condition on the state

This directly pins down where the state path starts

We need to pin down the co-state path

Pinning down the optimal path

We have effectively two ODEs, one for $\dot{\lambda}(t)$ and one for $\dot{x}(t)$

We need two constraints

Constraint 1: Initial condition on the state

This directly pins down where the state path starts

We need to pin down the co-state path

We do this using **transverality conditions**

Pinning down the optimal path

In general there are four types of transversality conditions

Pinning down the optimal path

In general there are four types of transversality conditions

The first two are for free initial or terminal time problems: these are problems where the agent can select when the problem starts or ends

Pinning down the optimal path

In general there are four types of transversality conditions

The first two are for free initial or terminal time problems: these are problems where the agent can select when the problem starts or ends

The second two are for pinning down the initial and terminal state variables if they're free

Pinning down the optimal path

In general there are four types of transversality conditions

The first two are for free initial or terminal time problems: these are problems where the agent can select when the problem starts or ends

The second two are for pinning down the initial and terminal state variables if they're free

Usually terminal conditions are free and initial conditions are not

Pinning down the optimal path example

If the terminal state is free, the transversality condition is that its shadow value must be zero

Pinning down the optimal path example

If the terminal state is free, the transversality condition is that its shadow value must be zero

Why?

Pinning down the optimal path example

If the terminal state is free, the transversality condition is that its shadow value must be zero

Why?

If it were positive the policymaker could profitably deviate by altering the level of the stock. Finally, these are all necessary conditions of the problem

Discounting

We discount the future using exponential discounting $\rightarrow$ now time can directly affect value

Discounting

We discount the future using exponential discounting $\rightarrow$ now time can directly affect value

Assume that time does not directly affect instantaneous payoffs or the transitions equations

Discounting

We discount the future using exponential discounting $\rightarrow$ now time can directly affect value

Assume that time does not directly affect instantaneous payoffs or the transitions equations

Then our value is $J(x(t), u(t), t) = e^{-rt} V(x(t), u(t))$

Discounting

We discount the future using exponential discounting $\rightarrow$ now time can directly affect value

Assume that time does not directly affect instantaneous payoffs or the transitions equations

Then our value is $J(x(t), u(t), t) = e^{-rt}\, V(x(t), u(t))$

J yields the present, time 0 value of the change in value at time t

Discounting

We discount the future using exponential discounting $\rightarrow$ now time can directly affect value

Assume that time does not directly affect instantaneous payoffs or the transitions equations

Then our value is $J(x(t), u(t), t) = e^{-rt}\, V(x(t), u(t))$

J yields the present, time 0 value of the change in value at time t

J is the *present value* and V is the *current value*

Discounting

We discount the future using exponential discounting $\rightarrow$ now time can directly affect value

Assume that time does not directly affect instantaneous payoffs or the transitions equations

Then our value is $J(x(t), u(t), t) = e^{-rt} V(x(t), u(t))$

J yields the present, time 0 value of the change in value at time t

J is the *present value* and V is the *current value*

Present value refers to the value with respect to a specific period that we call the present

Current value terms

Our previous necessary conditions apply to present value Hamiltonians,
but let us analyze a current value Hamiltonian to avoid including time terms,

$$\begin{aligned} H^{cv}(x(t), u(t), \mu(t)) &\equiv e^{rt} H(x(t), u(t), \lambda(t), t) \\ &= e^{rt} J(x(t), u(t), t) + e^{rt}\lambda(t) g(x(t), u(t)) \end{aligned}$$

$\mu(t)$ is the shadow value λ brought into current value terms: $\mu(t) = e^{rt}\lambda(t)$

Current value terms

We can then re-write our necessary conditions in current value by substituting in for:

- the shadow value (which implies that $\dot{\lambda}(t) = -re^{-rt}\mu(t) + e^{-rt}\dot{\mu}(t)$)
- $\partial H/\partial x = e^{-rt}\,\partial H^{cv}/\partial x$ into our co-state condition:

$$e^{-rt}\,\frac{\partial H^{cv}(x(t), u(t), \mu(t))}{\partial x} = e^{-rt}\left[r\mu(t) - \dot{\mu}(t)\right]$$

Current value

$$e^{-rt}\,\frac{\partial H^{cv}(x(t),u(t),\mu(t))}{\partial x} = e^{-rt}\left[r\mu(t) - \dot{\mu}(t)\right]$$

Current value

$$e^{-rt}\,\frac{\partial H^{cv}(x(t),u(t),\mu(t))}{\partial x} = e^{-rt}\left[r\mu(t) - \dot{\mu}(t)\right]$$

Before, the present value form of the co-state condition required the change in the present shadow value precisely equal the effect of the state variable on instantaneous value

Current value

$$e^{-rt}\,\frac{\partial H^{cv}(x(t),u(t),\mu(t))}{\partial x} = e^{-rt}\left[r\mu(t) - \dot{\mu}(t)\right]$$

In current value form, the co-state condition recognizes that the change in the present shadow value is comprised of two parts:

1. The change in the current shadow value
2. The reduction in present value purely from the passage of time

Current value

$$e^{-rt}\frac{\partial H^{cv}(x(t),u(t),\mu(t))}{\partial x} = e^{-rt}\left[r\mu(t) - \dot{\mu}(t)\right]$$

In current value form, the co-state condition recognizes that the change in the present shadow value is comprised of two parts:

1. The change in the current shadow value
2. The reduction in present value purely from the passage of time

If discounting is high (large r), then the current shadow value must change quicker in order to compensate the policymaker for leaving stock for the future

Numerical methods for continuous time models

Continuous time models are systems of ODEs

Numerical methods for continuous time models

Continuous time models are systems of ODEs

A first-order ordinary differential equation has the form

$$dy/dt = f(y,t)$$

Numerical methods for continuous time models

Continuous time models are systems of ODEs

A first-order ordinary differential equation has the form

$$dy/dt = f(y, t)$$

Where $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$

Numerical methods for continuous time models

Continuous time models are systems of ODEs

A first-order ordinary differential equation has the form

$$dy/dt = f(y, t)$$

Where $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$

The unknown is the function $y(t) : [t_0, T] \to \mathbb{R}^n$

Numerical methods for continuous time models

Continuous time models are systems of ODEs

A first-order ordinary differential equation has the form

$$dy/dt = f(y, t)$$

Where $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$

The unknown is the function $y(t) : [t_0, T] \to \mathbb{R}^n$

n determines the number of differential equations that we have

Numerical methods for continuous time models

The differential equation will give us the shape of the path that is the solution, but not where that path lies in our state space

Numerical methods for continuous time models

The differential equation will give us the shape of the path that is the solution, but not where that path lies in our state space

We need additional conditions to pin down $y(t)$, how we pin down $y(t)$ is what defines the different types of problems we have

Numerical methods for continuous time models

The differential equation will give us the shape of the path that is the solution, but not where that path lies in our state space

We need additional conditions to pin down $y(t)$, how we pin down $y(t)$ is what defines the different types of problems we have

If we pin down $y(t_0) = y_0$ or $y(T) = y_T$ we have an initial value problem

Numerical methods for continuous time models

The differential equation will give us the shape of the path that is the solution, but not where that path lies in our state space

We need additional conditions to pin down $y(t)$, how we pin down $y(t)$ is what defines the different types of problems we have

If we pin down $y(t_0) = y_0$ or $y(T) = y_T$ we have an initial value problem

IVPs are defined by the function being pinned down at one end or the other

Numerical methods for continuous time models

In one dimension we must pin down the function with either an initial or terminal condition so they are all IVPs by default

Numerical methods for continuous time models

In one dimension we must pin down the function with either an initial or terminal condition so they are all IVPs by default

If $n > 1$ then we can have a *boundary value problem* where we impose n conditions on y

$$g_i(y(t_0)) = 0, \quad i = 1, \ldots, .n',$$
$$g_i(y(T)) = 0, \quad i = n' + 1, \ldots, n$$

where $g : \mathbb{R}^n \to \mathbb{R}^n$

Numerical methods for continuous time models

In general we have that

$$g_i(y(t_i)) = 0$$

for some set of points $t_i, t_0 \leq t_i \leq T, 1 \leq i \leq n$

Numerical methods for continuous time models

In general we have that

$$g_i(y(t_i)) = 0$$

for some set of points $t_i, t_0 \leq t_i \leq T, 1 \leq i \leq n$

Often we set $T = \infty$ so we will need some condition in the limit: $\lim_{t\to\infty} y(t)$

Numerical methods for continuous time models

Note two more things:

1. We are implicitly assuming that these n conditions are independent, otherwise we will not have a unique solution
2. IVPs and BVP are fundamentally different: IVPs are problems where the auxiliary conditions that pin down the solution are all at one point, in BVPs they can be at different points, this has significant implications for how we can solve the problems

Numerical methods for continuous time models

If we have higher-order ODEs we can use a simple change of variables

$$d^2y/dx^2 = g(dy/dx, y, x)$$

for $x, y \in \mathbb{R}$

Numerical methods for continuous time models

If we have higher-order ODEs we can use a simple change of variables

$$d^2y/dx^2 = g(dy/dx, y, x)$$

for $x, y \in \mathbb{R}$

Define $z = dy/dx$ and then we can study the alternative system

$$dy/dx = z \qquad dz/dx = g(z, y, x)$$

of two first-order ODEs

Numerical methods for continuous time models

If we have higher-order ODEs we can use a simple change of variables

$$d^2y/dx^2 = g(dy/dx, y, x)$$

for $x, y \in \mathbb{R}$

Define $z = dy/dx$ and then we can study the alternative system

$$dy/dx = z \qquad dz/dx = g(z, y, x)$$

of two first-order ODEs

In general you can always transform a nth-order ODE into n first-order ODEs

Finite difference methods for IVPs

We solve IVPs using finite-difference methods

Finite difference methods for IVPs

We solve IVPs using finite-difference methods

Consider the following IVP

$$y' = f(t, y), \qquad y(t_0) = y_0$$

Finite difference methods for IVPs

We solve IVPs using finite-difference methods

Consider the following IVP

$$y' = f(t, y), \qquad y(t_0) = y_0$$

A finite-difference method solves this IVP by first specifying a grid/mesh over t: $t_0 < t_1 < \ldots < t_i < \ldots$

Finite difference methods for IVPs

We solve IVPs using finite-difference methods

Consider the following IVP

$$y' = f(t, y), \qquad y(t_0) = y_0$$

A finite-difference method solves this IVP by first specifying a grid/mesh over t: $t_0 < t_1 < \ldots < t_i < \ldots$

Assume the grid is uniformly spaced: $t_i = t_0 + ih,\ i = 0, 1, \ldots, N$ where h is the mesh size

Finite difference methods for IVPs

We solve IVPs using finite-difference methods

Consider the following IVP

$$y' = f(t, y), \qquad y(t_0) = y_0$$

A finite-difference method solves this IVP by first specifying a grid/mesh over t: $t_0 < t_1 < \ldots < t_i < \ldots$

Assume the grid is uniformly spaced: $t_i = t_0 + ih,\ i = 0, 1, \ldots, N$ where h is the mesh size

Our goal is to find for each t_i, a value Y_i that closely approximates $y(t_i)$

Finite difference methods for IVPs

To do this, we replace our differential equation with a difference system on the grid

Finite difference methods for IVPs

To do this, we replace our differential equation with a difference system on the grid

For example we might have $Y_{i+1} = F(Y_i, Y_{i-1}, \dots, t_{i+1}, t_i, \dots)$

Finite difference methods for IVPs

To do this, we replace our differential equation with a difference system on the grid

For example we might have $Y_{i+1} = F(Y_i, Y_{i-1}, \ldots, t_{i+1}, t_i, \ldots)$

We then solve the difference equation for the Y's in sequence where the initial Y_0 is fixed by the initial condition $Y_0 = y_0$

Finite difference methods for IVPs

To do this, we replace our differential equation with a difference system on the grid

For example we might have $Y_{i+1} = F(Y_i, Y_{i-1}, \ldots, t_{i+1}, t_i, \ldots)$

We then solve the difference equation for the Y's in sequence where the initial Y_0 is fixed by the initial condition $Y_0 = y_0$

This approximates the solution only at the grid points, but then we can interpolate using standard procedures to get the approximate solution off the grid points

Euler's method

The workhorse finite-difference method is Euler's method

Euler's method

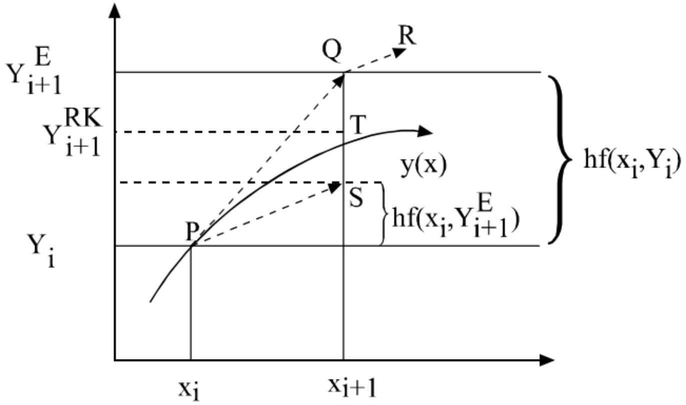
The workhorse finite-difference method is Euler's method

Euler's method is the difference equation

$$Y_{i+1} = Y_i + hf(t_i, Y_i)$$

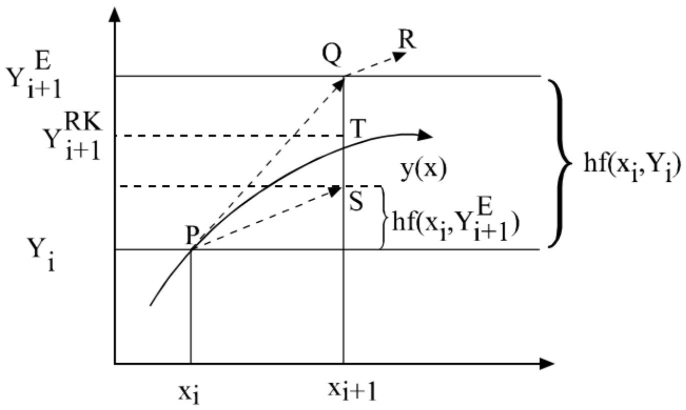
where Y_0 is given by the initial condition

Euler's method



Suppose the current iterate is $P = (t_i, Y_i)$ and $y(t)$ is the true solution

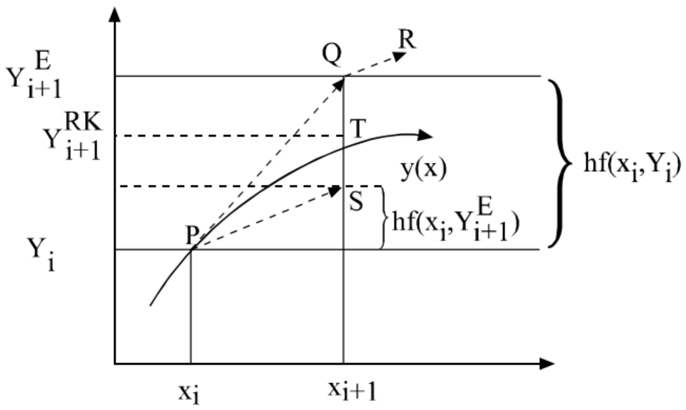
Euler's method



Suppose the current iterate is $P = (t_i, Y_i)$ and $y(t)$ is the true solution

At P, $y'(t_i)$ is the tangent vector $\vec{PQ}$

Euler's method

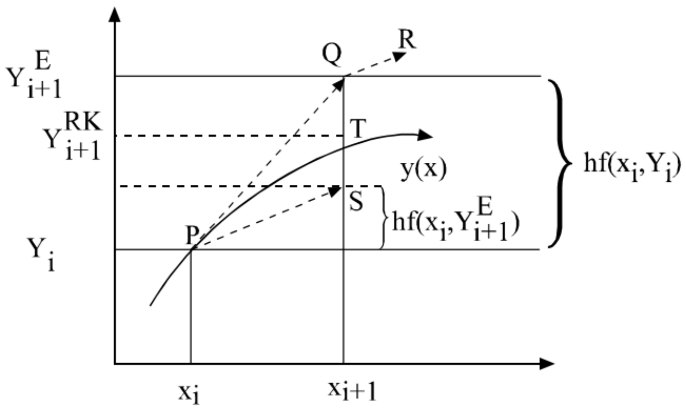


Suppose the current iterate is $P = (t_i, Y_i)$ and $y(t)$ is the true solution

At $P, y'(t_i)$ is the tangent vector $\vec{PQ}$

Euler's method follows that direction until $t = t_{i+1}$ at Q

Euler's method



Suppose the current iterate is $P = (t_i, Y_i)$ and $y(t)$ is the true solution

At $P, y'(t_i)$ is the tangent vector $\vec{PQ}$

Euler's method follows that direction until $t = t_{i+1}$ at Q

The Euler estimate of $y(t_{i+1})$ is then Y_{i+1}^E

Euler's method

This sounds very similar to Newton's method, because it is

Euler's method

This sounds very similar to Newton's method, because it is

Euler's method can be motivated by a similar Taylor approximation argument

Euler's method

This sounds very similar to Newton's method, because it is

Euler's method can be motivated by a similar Taylor approximation argument

If $y(t)$ is the true solution, the second order Taylor expansion around t_i is

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(\xi)$$

for some $\xi \in [t_i, t_{i+1}]$

Euler's method

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(\xi)$$

If we drop the second order term and assume $f(t_i, Y_i) = y'(t_i)$ and $Y_i = y(t_i)$ we have exactly Euler's formula

Euler's method

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(\xi)$$

If we drop the second order term and assume $f(t_i, Y_i) = y'(t_i)$ and $Y_i = y(t_i)$ we have exactly Euler's formula

For small $h, y(x)$ should be a close approximation to the solution of the truncated Taylor expansion,so Y_i should be a good approximation to $y(t_i)$

Euler's method

This approach approximated $y(t)$ with a linear function with slope $f(t_i, Y_i)$ on the interval $[t_i, t_{i+1}]$

Euler's method

This approach approximated $y(t)$ with a linear function with slope $f(t_i, Y_i)$ on the interval $[t_i, t_{i+1}]$

We can motivate Euler's method with an integration argument instead of a Taylor expansion argument

Euler's method

This approach approximated $y(t)$ with a linear function with slope $f(t_i, Y_i)$ on the interval $[t_i, t_{i+1}]$

We can motivate Euler's method with an integration argument instead of a Taylor expansion argument

The fundamental theorem of calculus tells us that

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(s, y(s))ds$$

Euler's method

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(s, y(s))ds$$

If we approximate the integral with $hf(t_i, y(t_i))$, a box of width h and height $f(t_i, y(t_i))$, then $y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i))$ which implies the Euler method difference equation above if $Y_i = y(t_i)$

Euler's method

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(s, y(s))ds$$

If we approximate the integral with $hf(t_i, y(t_i))$, a box of width h and height $f(t_i, y(t_i))$, then $y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i))$ which implies the Euler method difference equation above if $Y_i = y(t_i)$

Thus this also approximate $y(t)$ with a linear function over each subinterval with slope $f(t_i, Y_i)$

Euler's method

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(s, y(s))ds$$

If we approximate the integral with $hf(t_i, y(t_i))$, a box of width h and height $f(t_i, y(t_i))$, then $y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i))$ which implies the Euler method difference equation above if $Y_i = y(t_i)$

Thus this also approximate $y(t)$ with a linear function over each subinterval with slope $f(t_i, Y_i)$

As h decreases, we would expect the solutions to become more accurate

Euler's method

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(s, y(s))ds$$

If we approximate the integral with $hf(t_i, y(t_i))$, a box of width h and height $f(t_i, y(t_i))$, then $y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i))$ which implies the Euler method difference equation above if $Y_i = y(t_i)$

Thus this also approximate $y(t)$ with a linear function over each subinterval with slope $f(t_i, Y_i)$

As h decreases, we would expect the solutions to become more accurate

As $h \to 0$, we are back in the ODE world

Euler's method errors

Consider a system, $y'(t) = y(t), y(0) = 1$

Euler's method errors

Consider a system, $y'(t) = y(t), y(0) = 1$

The solution to this is simply $y(t) = e^t$

Euler's method errors

Consider a system, $y'(t) = y(t), y(0) = 1$

The solution to this is simply $y(t) = e^t$

The Euler method gives us a difference equation of
$Y_{i+1} = Y_i + hY_i = (1 + h)Y_i$

Euler's method errors

Consider a system, $y'(t) = y(t), y(0) = 1$

The solution to this is simply $y(t) = e^t$

The Euler method gives us a difference equation of
$Y_{i+1} = Y_i + hY_i = (1+h)Y_i$

This difference equation has solution $Y_i = (1+h)^i$ and implies the approximation is $Y(t) = (1+h)^{t/h}$

Euler's method errors

Thus the relative error between the two is

$$\log(|Y(t)/y(t)|) = \frac{t}{h}\log(1+h) - t = \frac{t}{h}(h - h^2 + \dots) - t = -th + \dots$$

where excluded terms have order higher than h

Euler's method errors

Thus the relative error between the two is

$$\log(|Y(t)/y(t)|) = \frac{t}{h}\log(1+h) - t = \frac{t}{h}(h - h^2 + \dots) - t = -th + \dots$$

where excluded terms have order higher than h

Thus the relative error in the Euler approximation has order h and as h goes to zero so does the approximation error

Euler's method errors

In general we can show that Euler's method has linear convergence

Suppose the solution to $y'(t) = f(t, y(t)), y(t_0) = y_0$ *is* C^3 *on* $[t_0, T]$, *that* f *is* C^2 , *and that* f_y *and* f_{yy} *are bounded for all* y *and* $t_0 \leq t \leq T$. *Then the error of the Euler scheme with step size* h *is* $O(h)$

Euler's method code

Euler's method code

```
function euler_ode(df, t0, y0, h, n)

    t = zeros(n+1)
    y = zeros(n+1)

    # set the initial values
    t[1] = t0
    y[1] = y0

    # use Euler's method to approximate the solution at each step
    for i in 1:n
        t[i+1] = t[i] + h
        y[i+1] = y[i] + h * df(t[i], y[i])
    end

    return (t, y)

end
```

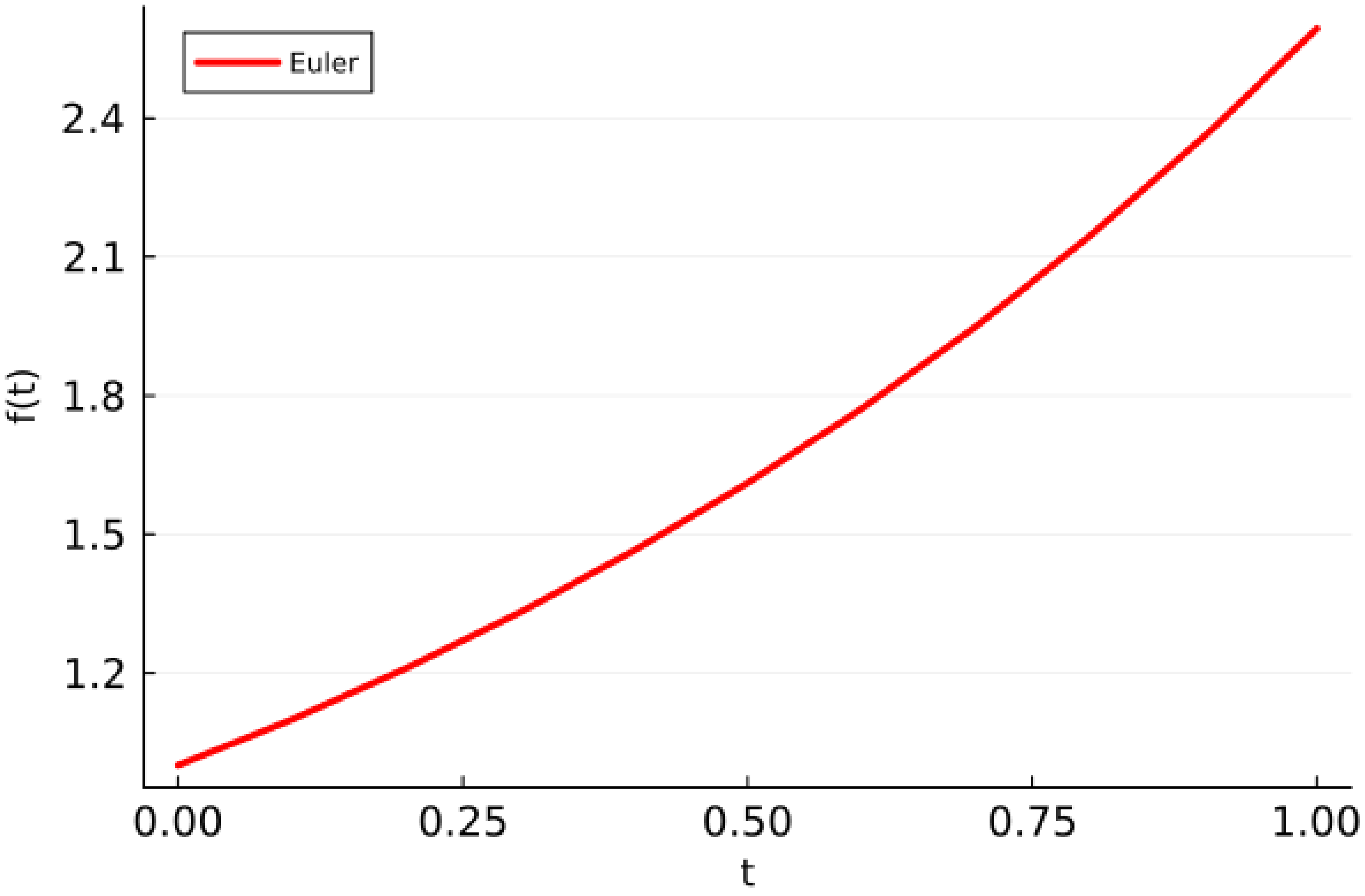
```
## euler_ode (generic function with 1 method)
```

Euler's method code

```
# dy/dt = y -> y = C_0*exp(t)
df(t, y) = y
t1, y1 = euler_ode(df, 0., 1., .1, 10)
```

Define df/dt and send it to the `euler_ode` function

Euler's method code



Implicit Euler method

We expanded y around t_i, but we could always expand around t_{i+1} so that we have

$$y(t_i) = y(t_{i+1}) - hy'(t_{i+1}) = y(t_{i+1}) - hf(t_{i+1}, y(t_{i+1}))$$

Implicit Euler method

We expanded y around t_i, but we could always expand around t_{i+1} so that we have

$$y(t_i) = y(t_{i+1}) - hy'(t_{i+1}) = y(t_{i+1}) - hf(t_{i+1}, y(t_{i+1}))$$

This yields the implicit Euler method

$$Y_{i+1} = Y_i + hf(t_{i+1}, Y_{i+1})$$

Implicit Euler method

We expanded y around t_i, but we could always expand around t_{i+1} so that we have

$$y(t_i) = y(t_{i+1}) - hy'(t_{i+1}) = y(t_{i+1}) - hf(t_{i+1}, y(t_{i+1}))$$

This yields the implicit Euler method

$$Y_{i+1} = Y_i + hf(t_{i+1}, Y_{i+1})$$

Notice that now Y_{i+1} is only implicitly defined in terms of t_i and Y_i so we will need to solve a non-linear equation in Y_{i+1}

Implicit Euler method

This seems not great, before we simply computed Y_{i+1} from values known at i but now we have to perform a rootfinding problem

Implicit Euler method

This seems not great, before we simply computed Y_{i+1} from values known at i but now we have to perform a rootfinding problem

However, Y_{i+1} does not simply depend on only Y_i and t_{i+1} but also f at t_{i+1}

Implicit Euler method

This seems not great, before we simply computed Y_{i+1} from values known at i but now we have to perform a rootfinding problem

However, Y_{i+1} does not simply depend on only Y_i and t_{i+1} but also f at t_{i+1}

Thus the implicit Euler method will get us better approximation properties, often times much better

Implicit Euler method

This seems not great, before we simply computed Y_{i+1} from values known at i but now we have to perform a rootfinding problem

However, Y_{i+1} does not simply depend on only Y_i and t_{i+1} but also f at t_{i+1}

Thus the implicit Euler method will get us better approximation properties, often times much better

Because of this we can typically use larger h's with the implicit Euler method

Implicit Euler's method code

```
# rootfinding portion of implicit Euler
function find_euler_root(df, y, t, h, y0, tol)

    y_new = y0
    y_old = y0
    error = Inf

    while error > tol
        y_new = y + h * df(t, y_new)
        error = abs((y_new - y_old)/y_old)
        y_old = deepcopy(y_new)
    end

    return y_new

end
```

```
## find_euler_root (generic function with 1 method)
```

Implicit Euler's method code

```
function euler_implicit_ode(df, t0, y0, h, n, tol = 1e-6)

    t = zeros(n+1)
    y = zeros(n+1)
    t[1] = t0
    y[1] = y0

    for i in 1:n
        t[i+1] = t[i] + h
        y[i+1] = find_euler_root(df, y[i], t[i+1], h, y[i], tol)
    end

    return (t, y)

end
```

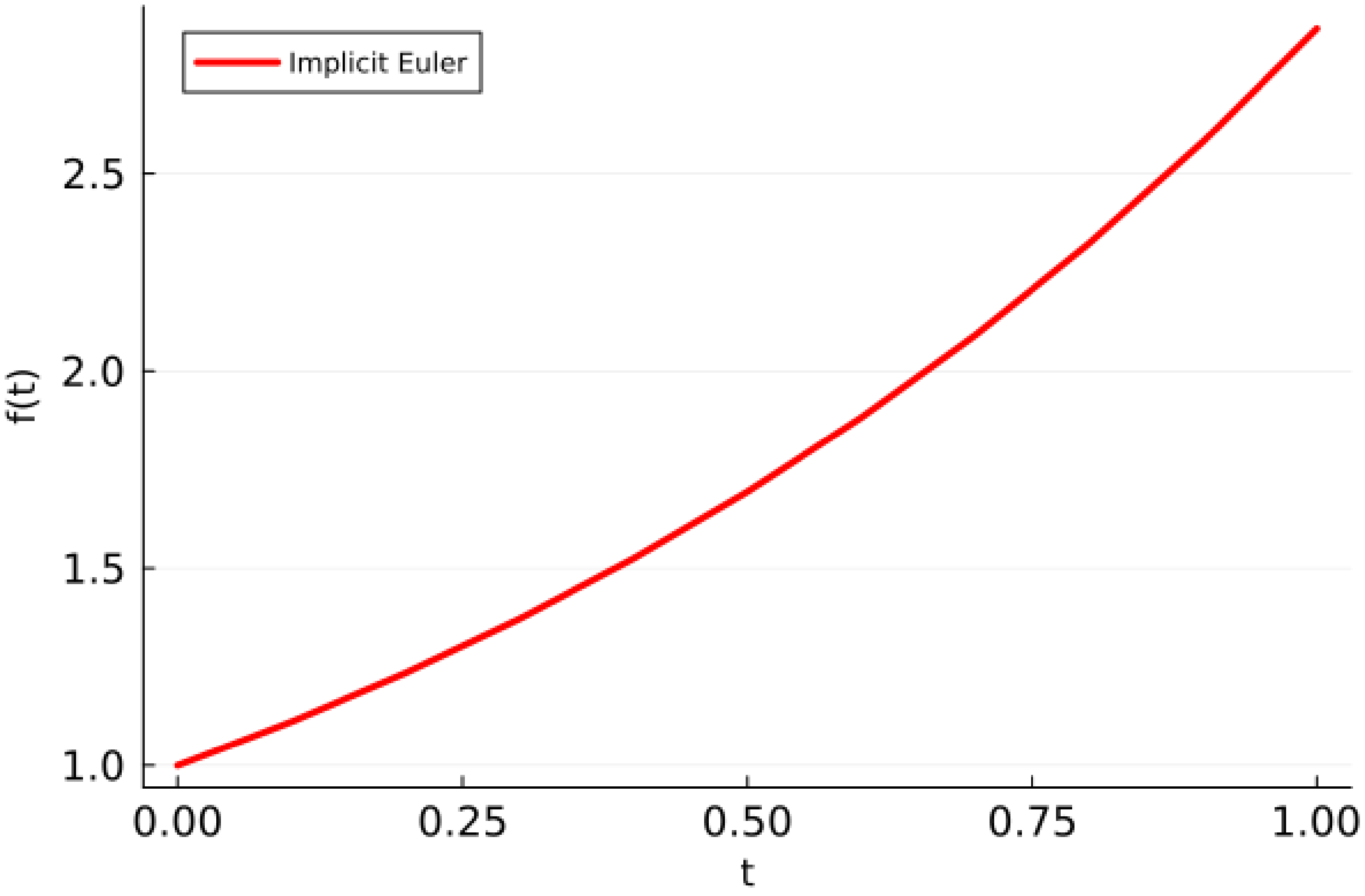
```
## euler_implicit_ode (generic function with 2 methods)
```

Implicit Euler's method code

```
df(t, y) = y
t2, y2 = euler_implicit_ode(df, 0., 1., .1, 10, 1e-6)
```

Define df/dt and send it to the `euler_implicit_ode` function

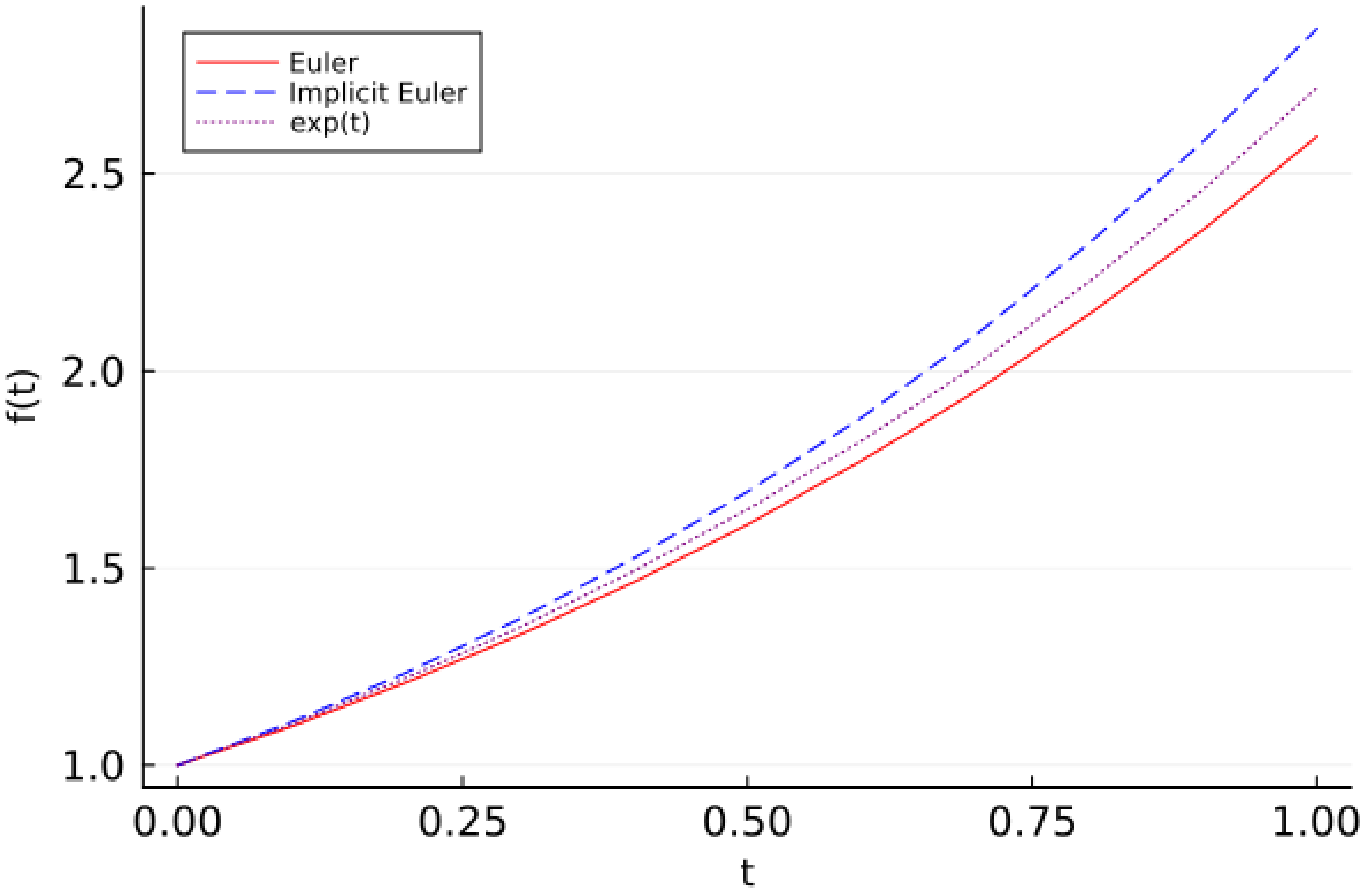
Implicit Euler's method code



Comparison

```
df(t, y) = y
t1, y1 = euler_ode(df, 0., 1., .1, 10)
t2, y2 = euler_implicit_ode(df, 0., 1., .1, 10, 1e-6)
y_real = exp.(t1)
```

Comparison



Runge-Kutta methods

Runge-Kutta methods take Euler methods but adapts f

Runge-Kutta methods

Runge-Kutta methods take Euler methods but adapts f

In the standard Euler approach we implicitly assume that the slope at (t_{i+1}, Y_{i+1}^E) is the same as the slope at (t_i, Y_i^E) which is a bad assumption unless y is linear

Runge-Kutta methods

Runge-Kutta methods take Euler methods but adapts f

In the standard Euler approach we implicitly assume that the slope at (t_{i+1}, Y_{i+1}^E) is the same as the slope at (t_i, Y_i^E) which is a bad assumption unless y is linear

For example if y is concave we will overshoot the true value

Runge-Kutta methods

Runge-Kutta methods take Euler methods but adapts f

In the standard Euler approach we implicitly assume that the slope at (t_{i+1}, Y_{i+1}^E) is the same as the slope at (t_i, Y_i^E) which is a bad assumption unless y is linear

For example if y is concave we will overshoot the true value

We could instead use the slope at (t_{i+1}, Y_{i+1}^E) but this will give the same problem but in the opposite direction, we will undershoot

Runge-Kutta methods

Runge-Kutta methods recognizes these two facts

Runge-Kutta methods

Runge-Kutta methods recognizes these two facts

A first-order Runge-Kutta method will take the average of these two slopes to arrive at the formula

$$Y_{i+1} = Y_i + \frac{h}{2}[f(t_i, Y_i) + f(t_{i+1}, Y_{i+1})]$$

Runge-Kutta methods

Runge-Kutta methods recognizes these two facts

A first-order Runge-Kutta method will take the average of these two slopes to arrive at the formula

$$Y_{i+1} = Y_i + \frac{h}{2}\left[f(t_i, Y_i) + f(t_{i+1}, Y_{i+1})\right]$$

This converges quadratically to the true solution in h, but now uses two evaluations per step

Runge-Kutta methods

Runge-Kutta methods recognizes these two facts

A first-order Runge-Kutta method will take the average of these two slopes to arrive at the formula

$$Y_{i+1} = Y_i + \frac{h}{2}[f(t_i, Y_i) + f(t_{i+1}, Y_{i+1})]$$

This converges quadratically to the true solution in h, but now uses two evaluations per step

This has the same flavor as forward vs central finite differences

Runge-Kutta methods

Runge-Kutta methods recognizes these two facts

A first-order Runge-Kutta method will take the average of these two slopes to arrive at the formula

$$Y_{i+1} = Y_i + \frac{h}{2}[f(t_i, Y_i) + f(t_{i+1}, Y_{i+1})]$$

This converges quadratically to the true solution in h, but now uses two evaluations per step

This has the same flavor as forward vs central finite differences

There are higher order Runge-Kutta rules that have even more desirable

Runge-Kutta code

```
function find_euler_root_rk(df, y, t, tp, h, y0, tol)

    y_new = y0
    y_old = y0
    error = Inf

    while error > tol
        y_new = y + h/2 * (df(t, y) + df(tp, y_new))
        error = abs((y_new - y_old)/y_old)
        y_old = deepcopy(y_new)
    end

    return y_new

end
```

```
## find_euler_root_rk (generic function with 1 method)
```

Runge-Kutta code

```
function euler_rk_ode(df, t0, y0, h, n, tol = 1e-6)

    t = zeros(n+1)
    y = zeros(n+1)
    t[1] = t0
    y[1] = y0

    for i in 1:n
        t[i+1] = t[i] + h
        y[i+1] = find_euler_root_rk(df, y[i], t[i], t[i+1], h, y0, tol)
    end

    return (t, y)

end
```

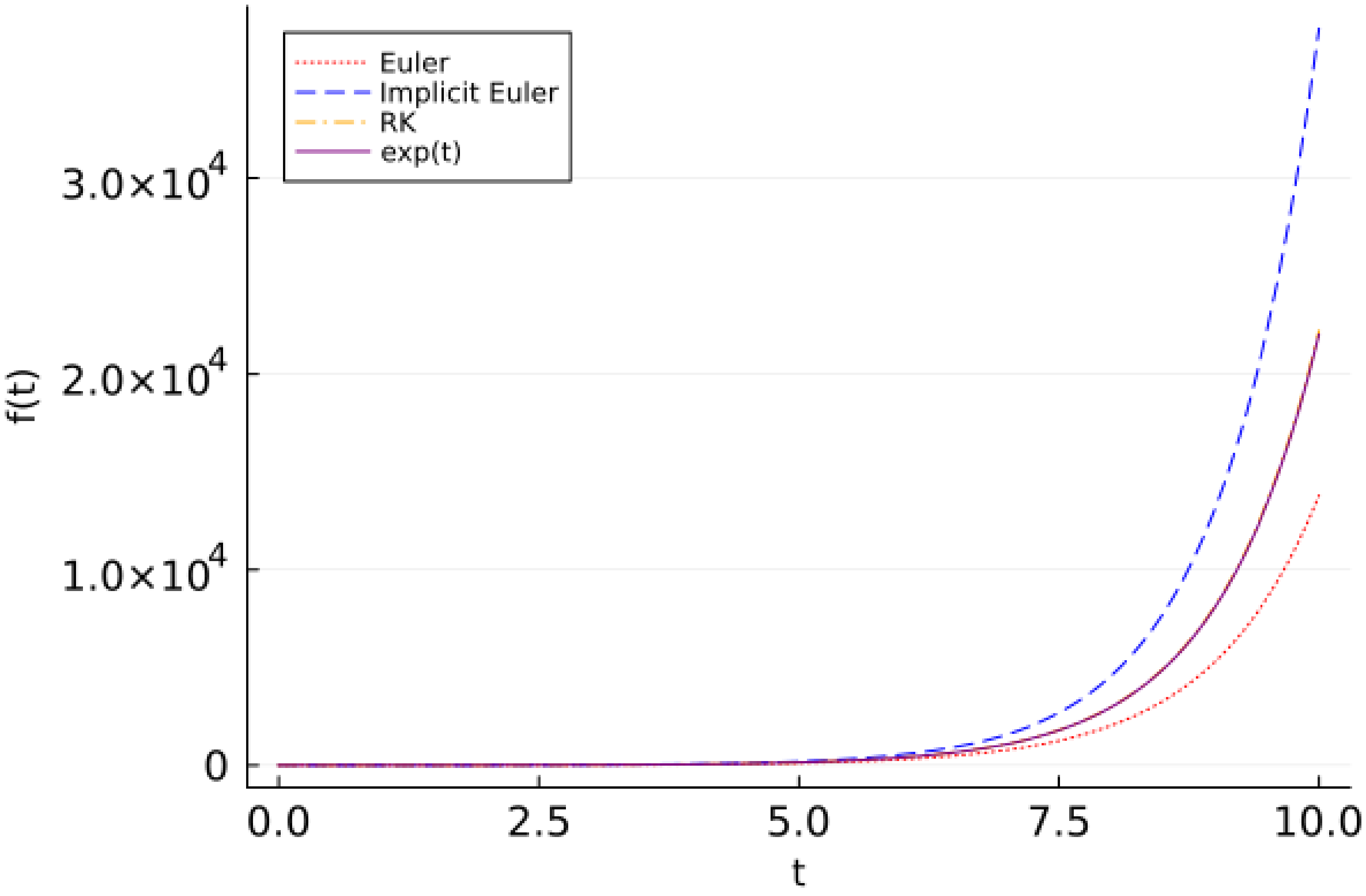
```
## euler_rk_ode (generic function with 2 methods)
```

Comparison

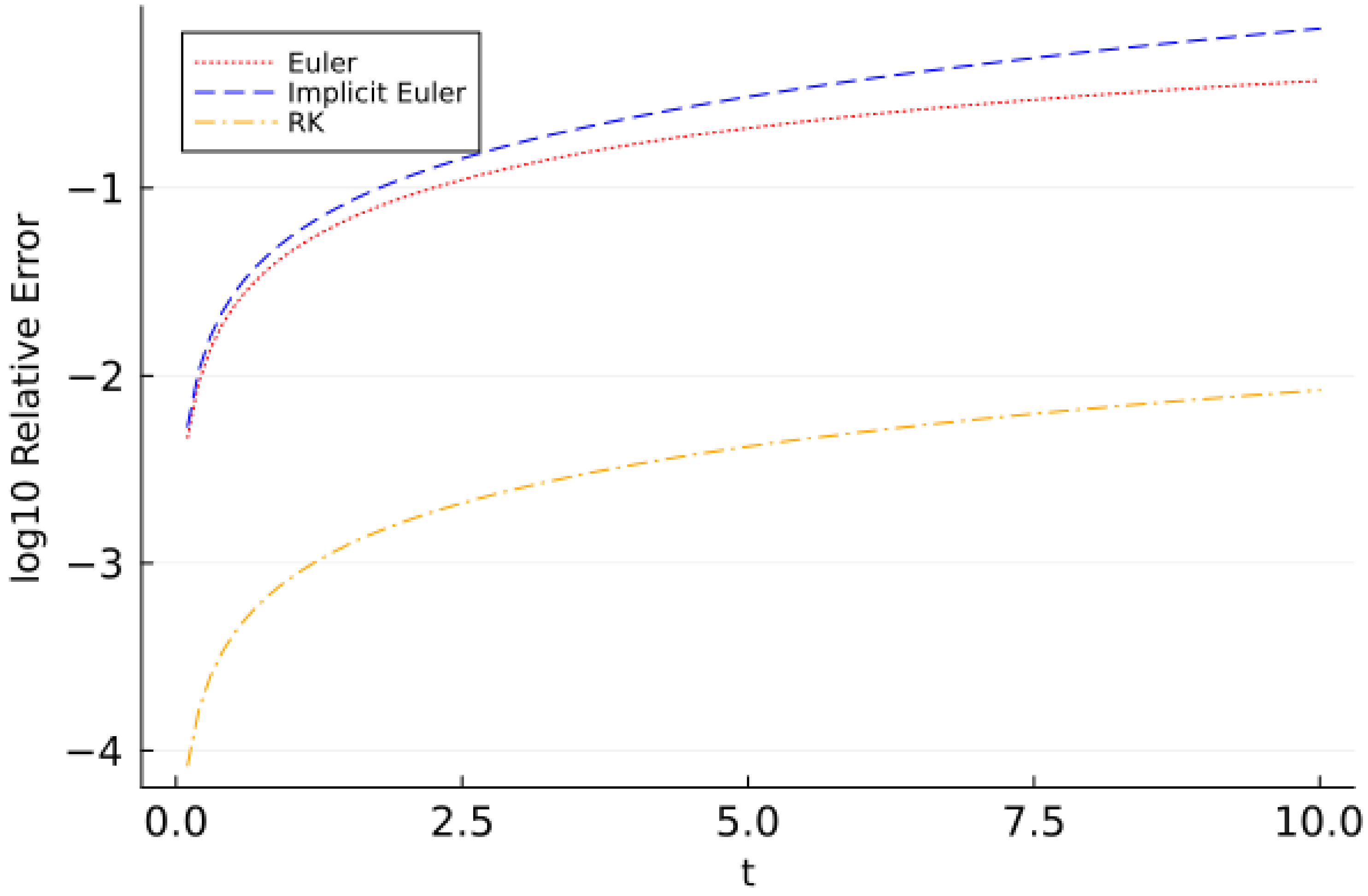
```
df(t, y) = y
t1, y1 = euler_ode(df, 0., 1., .1, 100)
t2, y2 = euler_implicit_ode(df, 0., 1., .1, 100, 1e-7)
t2, y3 = euler_rk_ode(df, 0., 1., .1, 100, 1e-7)
y_real = exp.(t1)
```

Check the time/memory with `@btime` in `BenchmarkTools`

Comparison: RK has minimal error



Comparison: RK has minimal error



Boundary Value Problems

IVPs are easy to solve because the solution depends only on local conditions so we can use local solution algorithms which are convenient

Boundary Value Problems

IVPs are easy to solve because the solution depends only on local conditions so we can use local solution algorithms which are convenient

BVPs have auxiliary conditions that are imposed at different points in time so we lose the local nature of the problem and our solutions must now be global in nature

Boundary Value Problems

Consider the following BVP

$$\begin{aligned} \dot{x} &= f(x, y, t) \\ \dot{y} &= g(x, y, t) \\ x(t_0) = x_0, \quad y(T) &= y_T \end{aligned}$$

where $x \in \mathbb{R}^n, y \in \mathbb{R}^m$

Boundary Value Problems

Consider the following BVP

$$\begin{aligned} \dot{x} &= f(x, y, t) \\ \dot{y} &= g(x, y, t) \\ x(t_0) = x_0, \quad y(T) &= y_T \end{aligned}$$

where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$

We cannot use standard IVP approaches because at t_0 or T we only know the value of either x or y but not both

Boundary Value Problems

Consider the following BVP

$$\begin{aligned} \dot{x} &= f(x, y, t) \\ \dot{y} &= g(x, y, t) \\ x(t_0) = x_0, \quad y(T) &= y_T \end{aligned}$$

where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$

We cannot use standard IVP approaches because at t_0 or T we only know the value of either x or y but not both

Thus we cannot find the next value of both of them using only local information: we need alternative approaches

Boundary Value Problems: Shooting

The core method for solving BVPs is called **shooting**

Boundary Value Problems: Shooting

The core method for solving BVPs is called **shooting**

The idea behind shooting is that we guess the value of $y(t_0)$, and use an IVP method to see what that means about $y(T)$

Boundary Value Problems: Shooting

The core method for solving BVPs is called **shooting**

The idea behind shooting is that we guess the value of $y(t_0)$, and use an IVP method to see what that means about $y(T)$

For any given guess, we generally won't hit the terminal condition exactly and might not even be that close

Boundary Value Problems: Shooting

The core method for solving BVPs is called **shooting**

The idea behind shooting is that we guess the value of $y(t_0)$, and use an IVP method to see what that means about $y(T)$

For any given guess, we generally won't hit the terminal condition exactly and might not even be that close

But we do get some information from where we end up at $y(T)$ and can use that information to update our guesses for $y(t_0)$ until we are sufficiently close to y_T

Boundary Value Problems: Shooting

The core method for solving BVPs is called **shooting**

The idea behind shooting is that we guess the value of $y(t_0)$, and use an IVP method to see what that means about $y(T)$

For any given guess, we generally won't hit the terminal condition exactly and might not even be that close

But we do get some information from where we end up at $y(T)$ and can use that information to update our guesses for $y(t_0)$ until we are sufficiently close to y_T

There are two components to a shooting method

Shooting

First we guess some $y(0) = y_0$ and then solve the IVP problem with methods we've already used

$$\begin{aligned} \dot{x} &= f(x, y, t) \\ \dot{y} &= g(x, y, t) \\ x(t_0) = x_0, \quad y(0) &= y_0 \end{aligned}$$

to find some $y(T)$ which we call $Y(T, y_0)$ since it depends on our initial guess y_0

Shooting

Second we need to find the *right* y_0

Shooting

Second we need to find the *right* y_0

We want to find a y_0 such that $y_T = Y(T, y_0)$

Shooting

Second we need to find the *right* y_0

We want to find a y_0 such that $y_T = Y(T, y_0)$

This is a nonlinear equation in y_0 so we need to solve nonlinear equations

Shooting

Second we need to find the *right* y_0

We want to find a y_0 such that $y_T = Y(T, y_0)$

This is a nonlinear equation in y_0 so we need to solve nonlinear equations

We can write the algorithm as

1. Initialize: Guess y_0^i. Choose a stopping criterion $\epsilon > 0$
2. Solve the IVP for $x(T), y(T)$ given the initial condition $y_0 = y_0^i$
3. If $||y(T) - y_T|| < \epsilon$, STOP. Else choose y_0^{i+1} based on the previous values of y and go back to step 1

Shooting

This is an example of a two layer algorithm

Shooting

This is an example of a two layer algorithm

The inner layer (step 1) uses an IVP method that solves $Y(T, y_0)$ for any y_0

Shooting

This is an example of a two layer algorithm

The inner layer (step 1) uses an IVP method that solves $Y(T, y_0)$ for any y_0

This can be Euler, Runge-Kutta or anything else

Shooting

This is an example of a two layer algorithm

The inner layer (step 1) uses an IVP method that solves $Y(T, y_0)$ for any y_0

This can be Euler, Runge-Kutta or anything else

In the outer layer (step 2) we solve the nonlinear equation $Y(T, y_0) = y_T$

Shooting

This is an example of a two layer algorithm

The inner layer (step 1) uses an IVP method that solves $Y(T, y_0)$ for any y_0

This can be Euler, Runge-Kutta or anything else

In the outer layer (step 2) we solve the nonlinear equation $Y(T, y_0) = y_T$

We can use any nonlinear solver here, typically we do this by defining a subroutine that computes $Y(T, y_0) - y_T$ as a function of y_0 and then sends that subroutine to a rootfinding program

Example: Lifecycle model

A simple lifecycle model is given by

$$\begin{aligned} \max_{c(t)} \int_0^T & e^{-rt} u(c(t)) dt \\ s.t. \quad \dot{A}(t) &= f(A(t)) + w(t) - c(t) \\ A(0) &= A(T) = 0. \end{aligned}$$

$u(c(t))$ is utility from consumption, $w(t)$ is the wage rate, $A(t)$ are assets and $f(A(t))$ is the return on invested assets

Example: Lifecycle model

A simple lifecycle model is given by

$$\begin{aligned} \max_{c(t)} \int_0^T & e^{-rt} u(c(t)) dt \\ s.t. \quad \dot{A}(t) &= f(A(t)) + w(t) - c(t) \\ A(0) &= A(T) = 0. \end{aligned}$$

$u(c(t))$ is utility from consumption, $w(t)$ is the wage rate, $A(t)$ are assets and $f(A(t))$ is the return on invested assets

We assume that assets are initially and terminally zero where the latter would come about naturally from a transversality condition

Example: Lifecycle model

The Hamiltonian is

$$H = u(c(t)) + \lambda(t)\left[f(A(t)) + w(t) - c(t)\right]$$

Example: Lifecycle model

The Hamiltonian is

$$H = u(c(t)) + \lambda(t)\left[f(A(t)) + w(t) - c(t)\right]$$

and the co-state condition is given by

$$\dot{\lambda}(t) = r\lambda(t) - \lambda(t)f'(A(t))$$

Example: Lifecycle model

The Hamiltonian is

$$H = u(c(t)) + \lambda(t)\left[f(A(t)) + w(t) - c(t)\right]$$

and the co-state condition is given by

$$\dot{\lambda}(t) = r\lambda(t) - \lambda(t)f'(A(t))$$

The maximum principle implies that $u'(c(t)) = \lambda(t)$

Example: Lifecycle model

This gives us a two equation system of differential equations (1 for the A transition, 1 for the costate condition) and the boundary conditions on A are what pin down the problem

Example: Lifecycle model

This gives us a two equation system of differential equations (1 for the A transition, 1 for the costate condition) and the boundary conditions on A are what pin down the problem

The issue here is that we never know A and λ at either $t = 0$ or $t = T$

Example: Lifecycle model

This gives us a two equation system of differential equations (1 for the A transition, 1 for the costate condition) and the boundary conditions on A are what pin down the problem

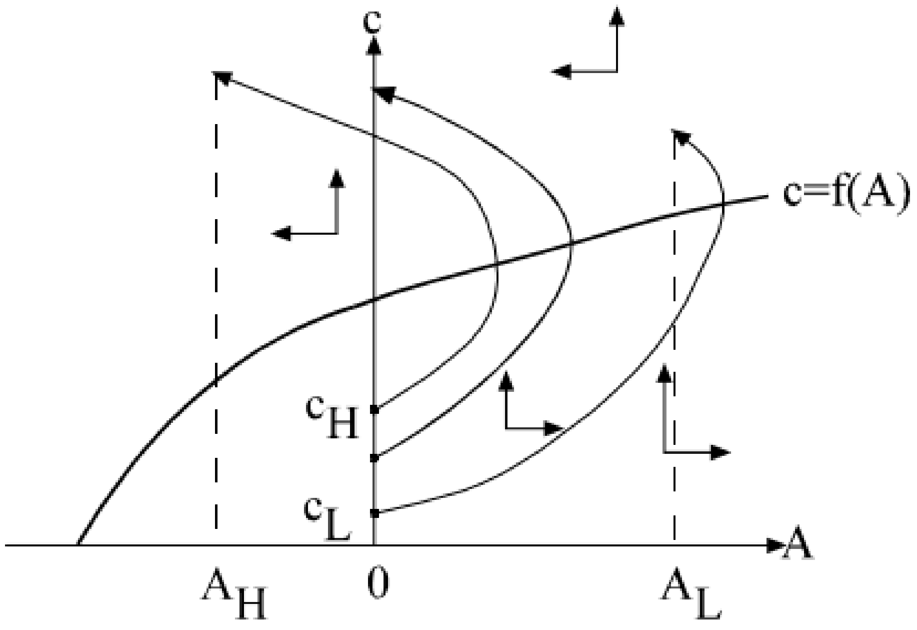
The issue here is that we never know A and λ at either $t = 0$ or $t = T$

We can use the maximum principle to convert the costate condition into a condition on consumption

$$\dot{c}(t) = -\frac{u'(c(t))}{u''(c(t))}\left[f'(A(t)) - r\right]$$

Example: Lifecycle model

The Figure shows the phase diagram assuming that $f'(A) > r$ for all A



If $A(T) < 0$ when we guess $c(0) = c_H$, but $A(T) > 0$ when we guess $c(0) = c_L$, we know the correct guess lies in between and we can solve for it using the

Example: Lifecycle model

Let's code it up

$$\dot{A}(t) = f(A(t)) + w(t) - c(t) \qquad \dot{c}(t) = -\frac{u'(c(t))}{u''(c(t))}\left[f'(A(t)) - r\right]$$

- $f(A(t)) = 1.05A(t)$
- $u(c(t)) = \log(c(t))$
- $w(t) = 5$
- $r = .02$

Example: Lifecycle model

Let's code it up

$$\dot{A}(t) = f(A(t)) + w(t) - c(t) \qquad \dot{c}(t) = -\frac{u'(c(t))}{u''(c(t))}\left[f'(A(t)) - r\right]$$

- $f(A(t)) = 1.05A(t)$
- $u(c(t)) = \log(c(t))$
- $w(t) = 5$
- $r = .02$

```
df(t,a,c) = (1.05*a + 5 - c, -1 * (1/c) / (-1/c^2) * (1.05 - .02))
```

```
## df (generic function with 2 methods)
```

Example: Lifecycle model

We need a 2 variable ODE solver next

Example: Lifecycle model

We need a 2 variable ODE solver next

```
function euler_ode(df, t0, a0, c0, h, n)

    t = zeros(n+1)
    a = zeros(n+1)
    c = zeros(n+1)

    t[1] = t0
    a[1] = a0
    c[1] = c0

    for i in 1:n
        t[i+1] = t[i] + h
        a[i+1] = a[i] + h * df(t[i], a[i], c[i])[1]
        c[i+1] = c[i] + h * df(t[i], a[i], c[i])[2]
    end

    return t, a, c

end
```

Example: Lifecycle model

Last, wrap it in bisection method

```
function solve_bvp(df, t0, a0, aend, c0low, c0high, h, n, tol = 1e-6)

    t = zeros(n+1)
    a = zeros(n+1)
    c = zeros(n+1)

    while abs.(c0low - c0high) > tol

        c0guess = (c0low + c0high)/2
        t, a, c = euler_ode(df, t0, a0, c0guess, h, n)
        anew = a[end]

        if sign(anew) > 0
            c0low = c0guess
        else
            c0high = c0guess
        end

    end
```

Example: Lifecycle model

Now we have to find the initial bounds, one where $A(T) > 0$, one where $A(T) < 0$

```
aend = 0.
a0 = 0.
t0 = 0.
h = .01
n = 100
c0low = 1     # low c0 guess
c0high = 10   # high c0 guess
```

Example: Lifecycle model

```
a0high = euler_ode(df, t0, a0, c0high, h, n)[2][end]
```

```
## -19.07974813179398
```

```
a0low = euler_ode(df, t0, a0, c0low, h, n)[2][end]
```

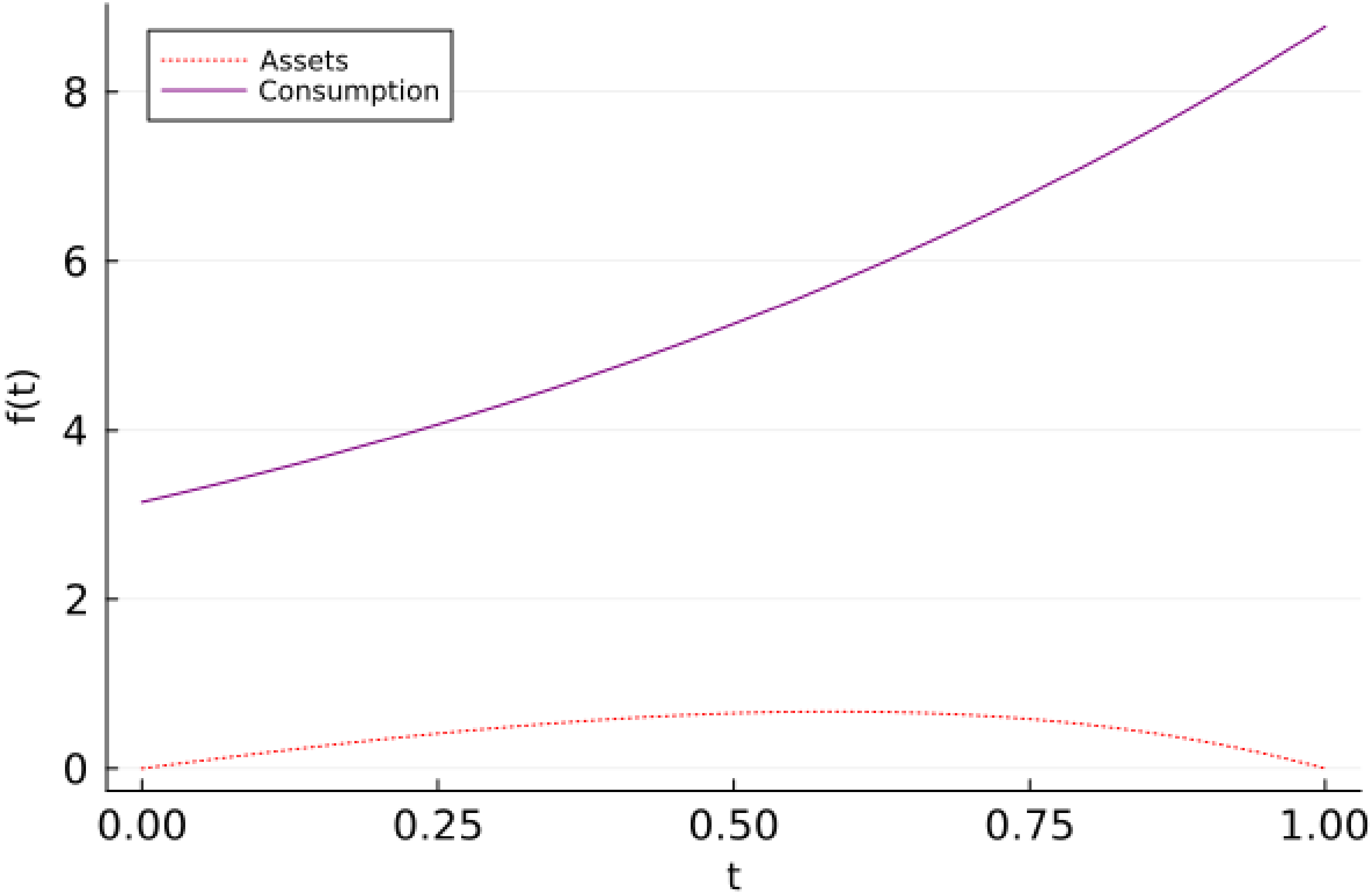
```
## 5.986527176940143
```

As expected, too high consumption $C(0)$ yields negative assets $A(T)$, too low consumption $C(0)$ yields positive assets $A(T)$

The $C(0)$ that solves the problem will fall somewhere in between

Example: Lifecycle model

```
t, a, c = solve_bvp(df, t0, a0, aend, c0low, c0high, h, n)
```



Reverse shooting for ∞ horizon problems

The standard infinite horizon optimal control problem is

$$\begin{aligned} \max_{u(t)} \int_0^\infty & e^{-rt}\pi(x(t), u(t))dt \\ s.t. \quad \dot{x}(t) &= f(x(t), u(t)) \\ x(0) &= x_0. \end{aligned}$$

Reverse shooting for ∞ horizon problems

The standard infinite horizon optimal control problem is

$$\begin{aligned} \max_{u(t)} \int_0^\infty & e^{-rt}\pi(x(t),u(t))dt \\ s.t. \quad \dot{x}(t) &= f(x(t),u(t)) \\ x(0) &= x_0. \end{aligned}$$

We still have $x(0) = x_0$ as before, but we no longer have the terminal condition

Reverse shooting for ∞ horizon problems

The standard infinite horizon optimal control problem is

$$
\begin{aligned}
\max_{u(t)} \int_0^\infty e^{-rt} \pi(x(t), u(t)) dt \\
s.t. \quad \dot{x}(t) = f(x(t), u(t)) \\
x(0) = x_0.
\end{aligned}
$$

We still have $x(0) = x_0$ as before, but we no longer have the terminal condition

We replace it with a transversality condition that $\lim_{t\to\infty} e^{-rt}|\lambda(t)^T x(t)| \leq \infty$

Reverse shooting for ∞ horizon problems

Shooting methods do not really work for infinite horizon problems since we need to integrate the problem over a very long time horizon and so $x(T)$ will be particularly sensitive to $\lambda(0)$ when T is large

Reverse shooting for ∞ horizon problems

Shooting methods do not really work for infinite horizon problems since we need to integrate the problem over a very long time horizon and so $x(T)$ will be particularly sensitive to $\lambda(0)$ when T is large

The primary issue is that the terminal state, because of the long time horizon, is very sensitive to the initial guess

Reverse shooting for ∞ horizon problems

Shooting methods do not really work for infinite horizon problems since we need to integrate the problem over a very long time horizon and so $x(T)$ will be particularly sensitive to $\lambda(0)$ when T is large

The primary issue is that the terminal state, because of the long time horizon, is very sensitive to the initial guess

But this implies something very convenient: that the initial state corresponding to any terminal state is not very sensitive to the value of the terminal state

Reverse shooting for ∞ horizon problems

Shooting methods do not really work for infinite horizon problems since we need to integrate the problem over a very long time horizon and so $x(T)$ will be particularly sensitive to $\lambda(0)$ when T is large

The primary issue is that the terminal state, because of the long time horizon, is very sensitive to the initial guess

But this implies something very convenient: that the initial state corresponding to any terminal state is not very sensitive to the value of the terminal state

Thus what we will do is not guess the value of the initial condition and integrate forward, we will guess the terminal condition and integrate

Example: Reverse shooting for ∞ horizon problems

Consider the simplest growth model

$$\begin{aligned} \max_{c(t)} \int_0^{\infty} e^{-rt} u(c(t)) dt \\ \dot{k}(t) = f(k(t)) - c(t) \\ s.t. \quad k(0) = k_0, \end{aligned}$$

where c is consumption, k is the capital stock, and f is production

Example: Reverse shooting for ∞ horizon problems

We can use Pontryagin's necessary conditions to get that consumption and capital are governed by the following differential equations

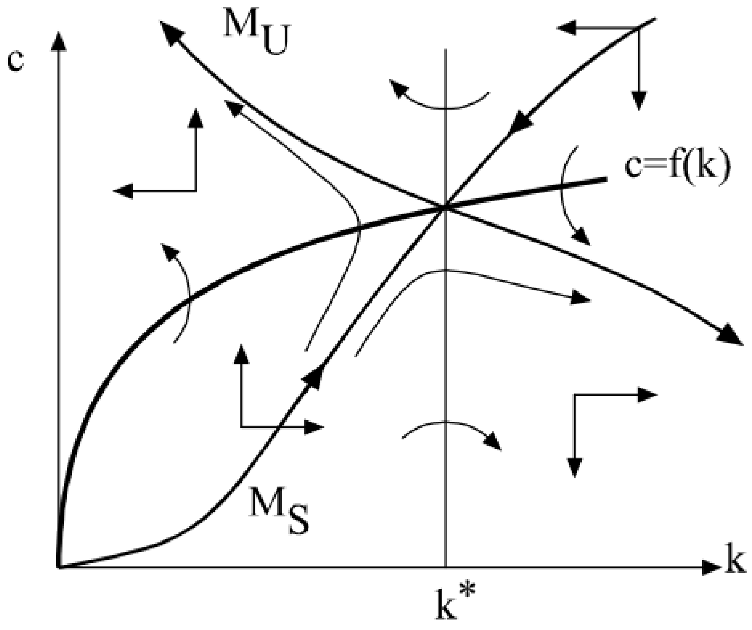
$$\dot{c}(t) = -\frac{u'(c(t))}{u''(c(t))}(f'(k) - r)$$
$$\dot{k}(t) = f(k(t)) - c(t),$$

with boundary conditions,

$$k(0) = k_0, \quad 0 < \lim_{t\to\infty} |k(t)| \leq \infty$$

Example: Reverse shooting for ∞ horizon problems

Assume u and f are concave, the Figure shows the phase diagram for the problem



We have a steady state at $k = k^*$, this occurs when $f'(k(t)) = r$ and

Example: Reverse shooting for ∞ horizon problems

For this problem there exists a stable manifold M_S and an unstable manifold M_U

so that the steady state is **saddle point stable**

Example: Reverse shooting for ∞ horizon problems

For this problem there exists a stable manifold M_S and an unstable manifold M_U
so that the steady state is **saddle point stable**

Both are invariant manifolds because any system that starts on either of these manifolds will continue to move along the manifold

Example: Reverse shooting for ∞ horizon problems

For this problem there exists a stable manifold M_S and an unstable manifold M_U
so that the steady state is **saddle point stable**

Both are invariant manifolds because any system that starts on either of these manifolds will continue to move along the manifold

However M_S is stable because it will converge to the steady state while M_U diverges away from the steady state

Example: Reverse shooting for ∞ horizon problems

Lets first use standard shooting to try to compute the stable manifold

Example: Reverse shooting for ∞ horizon problems

Lets first use standard shooting to try to compute the stable manifold

We want k and c to equal their steady state values at $t = \infty$, but we can't quite do that so we search for a $c(0)$ so that $(c(t), k(t))$ has a path that is close to the steady state

Example: Reverse shooting for ∞ horizon problems

Lets first use standard shooting to try to compute the stable manifold

We want k and c to equal their steady state values at $t = \infty$, but we can't quite do that so we search for a $c(0)$ so that $(c(t), k(t))$ has a path that is close to the steady state

Suppose we start with $k_0 < k^*$, if we guess $c(0)$ too big we will cross the k isoquant and have a falling capital stock, but if we guess $c(0)$ too small we will get a path that crosses the c isoquant and results in a falling consumption level

Example: Reverse shooting for ∞ horizon problems

This gives us our algorithm

1. Initialize: set $c_H = f(k_0)$ and set $c_L = 0$, choose a stopping criterion $\epsilon > 0$
2. Set $c_0 = \frac{1}{2}(c_L + c_H)$
3. Solve the IVP with initial conditions $c(0) = c_0, k(0) = k_0$. Stop the IVP at the first t when $\dot{c}(t) < 0$ or $\dot{k}(t) < 0$, denote this T
4. If $|c(T) - c^*| < \epsilon$, STOP. If $\dot{c}(t) < 0$, set $c_L = c_0$, else set $c_H = c_0$. Go to step 2.

Example: Reverse shooting for ∞ horizon problems

This algorithm makes sense but the phase diagram shows why it will have trouble finding the stable manifold

Example: Reverse shooting for ∞ horizon problems

This algorithm makes sense but the phase diagram shows why it will have trouble finding the stable manifold

Any small deviation from M_S is magnified and results in a path that increasingly gets far away from M_S

Example: Reverse shooting for ∞ horizon problems

This algorithm makes sense but the phase diagram shows why it will have trouble finding the stable manifold

Any small deviation from M_S is magnified and results in a path that increasingly gets far away from M_S

Unless we happen to pick a point precisely on the stable manifold we will move away from it, so it is hard to search for the solution since changes in our guesses will lead to wild changes in terminal values

Example: Reverse shooting for ∞ horizon problems

Now suppose we wanted to find a path on M_U, notice that the flow pushes points *toward* M_U so the deviations are smushed together

Example: Reverse shooting for ∞ horizon problems

Now suppose we wanted to find a path on M_U, notice that the flow pushes points *toward* M_U so the deviations are smushed together

If we wanted to compute a path that lies near the unstable manifold, we could simply pick a point near the steady state as the initial condition and integrate the system

Example: Reverse shooting for ∞ horizon problems

Now suppose we wanted to find a path on M_U, notice that the flow pushes points *toward* M_U so the deviations are smushed together

If we wanted to compute a path that lies near the unstable manifold, we could simply pick a point near the steady state as the initial condition and integrate the system

We don't actually want to solve for a path on M_U but this gives us some insight

Example: Reverse shooting for ∞ horizon problems

We want to change the system so that the stable manifold becomes the unstable manifold

Example: Reverse shooting for ∞ horizon problems

We want to change the system so that the stable manifold becomes the unstable manifold

We can do this easily by reversing time

$$\begin{aligned}\dot{c}(t) &= \frac{u'(c(t))}{u''(c(t))}(f'(k) - r) \\ \dot{k}(t) &= -(f(k(t)) - c(t))\end{aligned}$$

Example: Reverse shooting for ∞ horizon problems

We want to change the system so that the stable manifold becomes the unstable manifold

We can do this easily by reversing time

$$\dot{c}(t) = \frac{u'(c(t))}{u''(c(t))}(f'(k) - r)$$
$$\dot{k}(t) = -(f(k(t)) - c(t))$$

This yields the same phase diagram but with the arrows turned 180° so the stable manifold forward in time is now the unstable manifold reverse in time

Example: Reverse shooting for ∞ horizon problems

We want to change the system so that the stable manifold becomes the unstable manifold

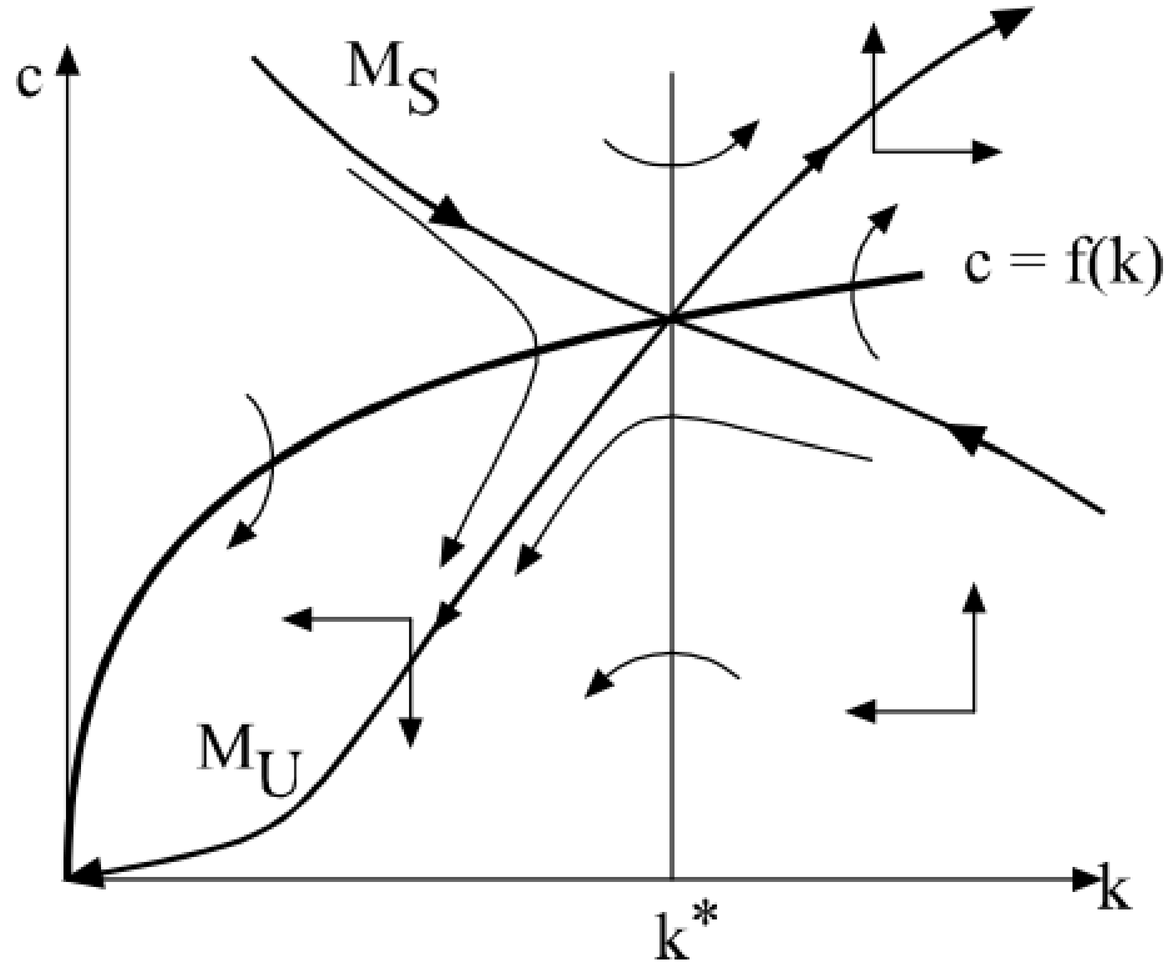
We can do this easily by reversing time

$$\dot{c}(t) = \frac{u'(c(t))}{u''(c(t))}(f'(k) - r)$$
$$\dot{k}(t) = -(f(k(t)) - c(t))$$

This yields the same phase diagram but with the arrows turned 180° so the stable manifold forward in time is now the unstable manifold reverse in time

This allows us to exploit how paths tend to converge toward the stable

Reverse shooting for ∞ horizon problems



Reverse shooting example

Let's code it up

- $f(k) = \sqrt{k}$
- $r = .02$
- $u(c) = log(c)$

Reverse shooting example

Let's code it up

- $f(k) = \sqrt{k}$
- $r = .02$
- $u(c) = log(c)$

We know the steady state is where $f'(k) = r$ and $f(k) = c$

Reverse shooting example

Let's code it up

- $f(k) = \sqrt{k}$
- $r = .02$
- $u(c) = log(c)$

We know the steady state is where $f'(k) = r$ and $f(k) = c$

This is $k = 625, c = 25$

Reverse shooting example

Let's code it up

- $f(k) = \sqrt{k}$
- $r = .02$
- $u(c) = log(c)$

We know the steady state is where $f'(k) = r$ and $f(k) = c$

This is $k = 625, c = 25$

Pretend we only knew $f(k) = c$ (from $\dot{K} = 0$) and solve by searching over terminal capital

Example: Lifecycle model

```
df(t,k,c) = (
  -(sqrt(k) - c),
  -(-1 * (1/c) / (-1/c^2) * (0.5*k^(-0.5) - .02))
  )
```

```
## df (generic function with 2 methods)
```

Reverse shooting example

We need a 2 variable ODE solver next

Reverse shooting example

We need a 2 variable ODE solver next

```
function euler_ode(df, t0, k0, c0, h, n)

    t = zeros(n+1)
    k = zeros(n+1)
    c = zeros(n+1)

    t[1] = t0
    k[1] = k0
    c[1] = c0

    for i in 1:n
        t[i+1] = t[i] + h
        k[i+1] = max(1e-6, k[i] + h * df(t[i], k[i], c[i])[1])
        c[i+1] = max(1e-6, c[i] + h * df(t[i], k[i], c[i])[2])
    end

    return t, k, c

end
```

Reverse shooting example

Last, wrap it in bisection method

```
function solve_bvp_rev(df, t0, k0, klow, khigh, h, n, tol = 1e-6)

    t = zeros(n+1)
    k = zeros(n+1)
    c = zeros(n+1)

    while abs.(klow - khigh) > tol
        kguess = (klow + khigh)/2
        t, k, c = euler_ode(df, t0, kguess, sqrt(kguess), h, n)
        anew = k[end]
        if anew < k0
            klow = kguess
        else
            khigh = kguess
        end
    end

    return t, k, c
```

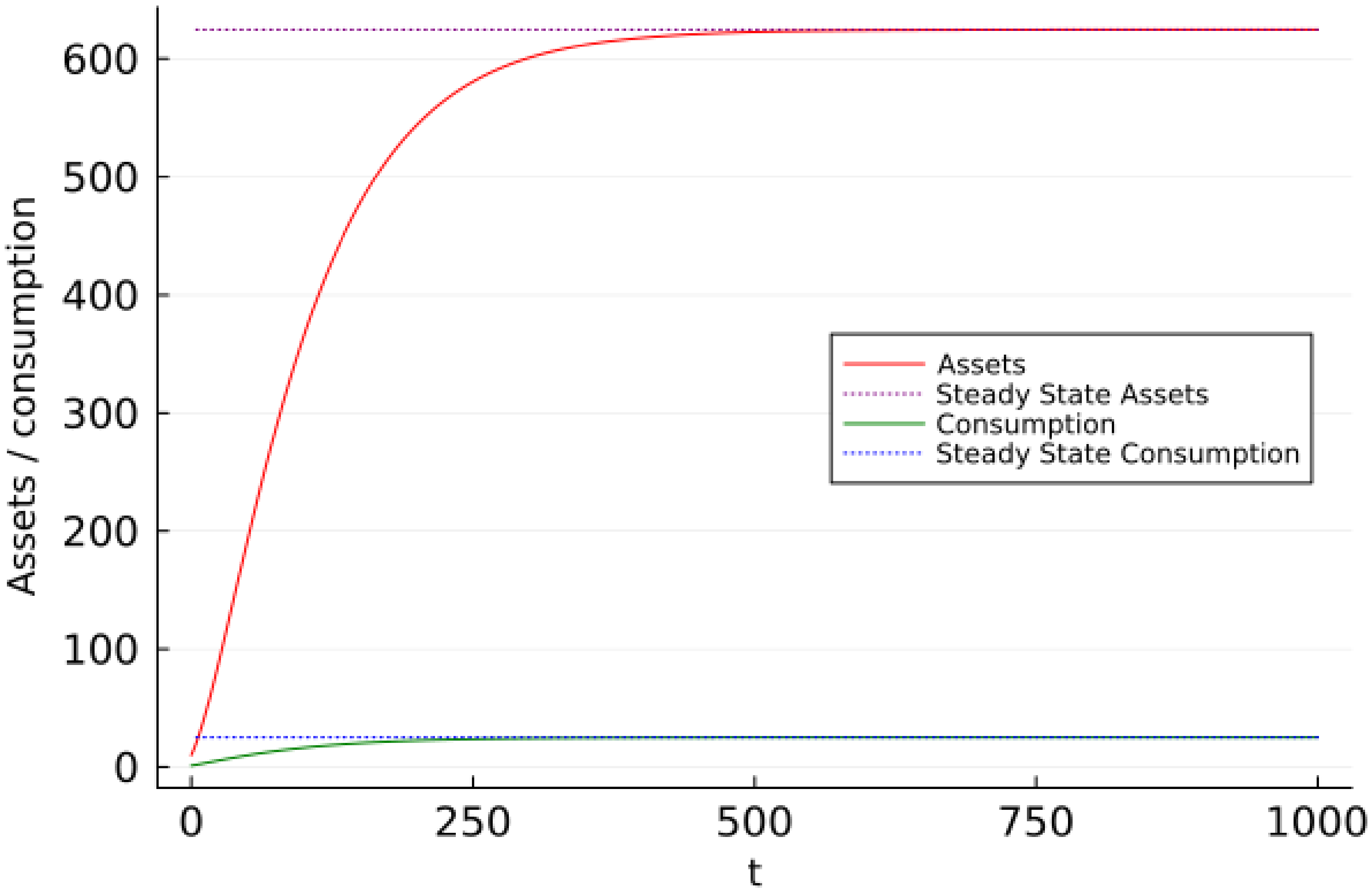
Reverse shooting example

Now we have to find the initial bounds, one where $A(T) > 0$, one where $A(T) < 0$

```
k0 = 10 # initial condition to hit
t0 = 0. # time starts at 0

klow = 100    # below closed-form solution of k = 625
khigh = 1000  # above closed-form solution of k = 625

aend = (.5 / .02)^2 # closed-form solution
cend = sqrt(aend) # closed-form solution

h = .1
n = 10000 # make the horizon long to approx infinite-horizon
```

Reverse shooting example

```
t, k, c = solve_bvp_rev(df, t0, k0, klow, khigh, h, n)
```



Reverse shooting example

```
k0 = 800
t, k, c = solve_bvp_rev(df, t0, k0, klow, khigh, h, n)
```