

Julia in Academia

TEXTBOOKS, STANFORD COURSES, AND THE FUTURE

ALGORITHMS FOR
VALIDATION

MYKEL J. KOCHENDERFER
SYDNEY M. KATZ
ANTHONY L. CORSO
ROBERT J. MOSS

92 CHAPTER 4. PREDICTION THROUGH OPTIMIZATION

Figure 4.3 A comparison of optimization-based localization on a 10x10 grid. The top row shows the true objective and the likelihood function for the first iteration of a population-based optimization algorithm, which will be discussed in the next section. The shaded gray path on the plate is the most likely path for the system. The bottom row shows the same algorithm that quickly moves towards the objective and a failure that stays close to the nominal path, only moving toward the outside at the end.

issues, other objective functions may lead to the discovery of more likely failure trajectories.

Another common objective for event likely failure analysis is

$$f(\mathbf{v}) = \rho(\mathbf{v}, \mathbf{g}) - \lambda \log(f(\mathbf{v})) \quad (4.8)$$

where λ is a weighting parameter selected by the user (Algorithm 4.6). This objectives is search and encourages the optimization algorithm to search simultaneously for trajectories that are both likely and close to failure.

```
function weighted_likelihood_objective(v, rho, w, w_likelihood, g, lambda)
    v = validate(v, g)
    if v == false
        return rho(v, g) + lambda * log(w(v))
    else
        w_likelihood(v) = w_likelihood(v) + log(w(v))
        return rho(v, g) - lambda * log(w_likelihood(v))
    end
end
```

Algorithm 4.6. Objective function that weighs the likelihood between robustness and likelihood. The function takes in a trajectory v , a system g , and a weighting parameter λ . It also takes in a likelihood function w and a rho function ρ . The function returns the weighted robustness plus the weighted likelihood. The likelihood function is set to 0, and the negative likelihood function is the nominal trajectory distribution.

4.6 Optimization Algorithms

We can search for failures by applying a variety of optimization algorithms to the optimization problems in equation (4.5).¹⁴ Algorithm 4.6 implements

© 2014 Kochenderfer, Katz, Corso, and Moss shared under a Creative Commons CC-BY-NC-ND license.
<http://www.cs.cmu.edu/~kochenderfer/>, kochenderfer@cs.cmu.edu

ROBERT MOSS, PHD
STANFORD UNIVERSITY | JULIACON 2025

mossr@cs.stanford.edu

WHO AM I

WHOAMI



Recent Stanford PhD Graduate
Used Julia throughout my research

WHOAMI



Prev. at MIT Lincoln Laboratory
Julia as a specification language

Recent Stanford PhD Graduate
Used Julia throughout my research

WHOAMI



Recent Stanford PhD Graduate
Used Julia throughout my research

A video thumbnail from JuliaCon 2015. It shows a man in a grey shirt and khaki pants standing at a podium, gesturing with his hands. The background includes a chalkboard and a wooden panel. The thumbnail has a blue border. Below the thumbnail, the text reads "Robert Moss" and "Using Julia as a Specification Language for the Next-Generation Airborne Collision Avoidance System".

Prev. at MIT Lincoln Laboratory
Julia as a specification language

ALGORITHMS FOR VALIDATION



MYKEL J. KOCHENDERFER
SYDNEY M. KATZ
ANTHONY L. CORSO
ROBERT J. MOSS

Textbook Co-Author/Head TA
Algorithms written in Julia

LONG-TIME JULIA USER

LONG-TIME JULIA USER

- Addict since 2013

LONG-TIME JULIA USER

- Addict since 2013
- Led Julia's use for aircraft collision avoidance systems at MIT-LL

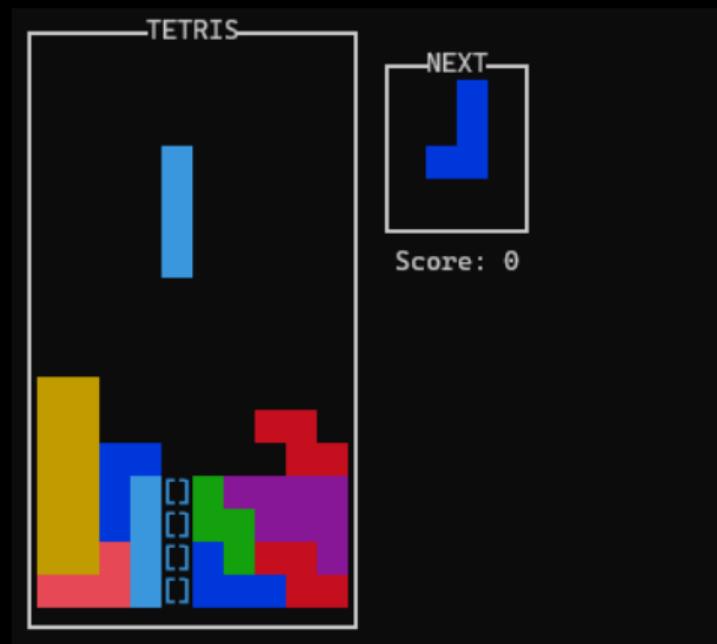
LONG-TIME JULIA USER

- Addict since 2013
- Led Julia's use for aircraft collision avoidance systems at MIT-LL
- Entire master's thesis and PhD dissertation research written in Julia

LONG-TIME JULIA USER

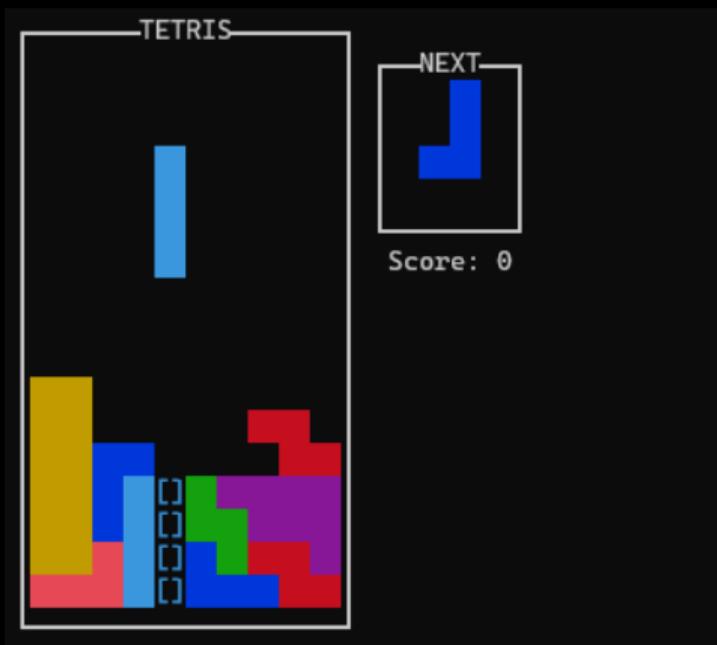
- Addict since 2013
- Led Julia's use for aircraft collision avoidance systems at MIT-LL
- Entire master's thesis and PhD dissertation research written in Julia
- Also enjoy some recreational coding and code golf

LONG-TIME JULIA USER



using `Tetris` in your REPL

LONG-TIME JULIA USER

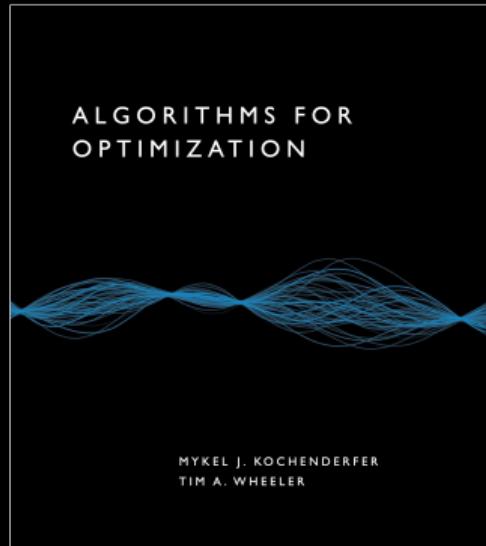


using Tetris in your REPL

One line of pure Julia

TEXTBOOKS USING JULIA

TEXTBOOKS USING JULIA



Algorithms for Optimization
MIT Press, 2019

TEXTBOOKS USING JULIA

ALGORITHMS FOR
OPTIMIZATION



MYKEL J. KOCHENDERFER
TIM A. WHEELER

ALGORITHMS FOR
DECISION MAKING



MYKEL J. KOCHENDERFER
TIM A. WHEELER
KYLE H. WRAY

Algorithms for Optimization
MIT Press, 2019

Algorithms for Decision Making
MIT Press, 2022

TEXTBOOKS USING JULIA

ALGORITHMS FOR
OPTIMIZATION



MYKEL J. KOCHENDERFER
TIM A. WHEELER

Algorithms for Optimization
MIT Press, 2019

ALGORITHMS FOR
DECISION MAKING



MYKEL J. KOCHENDERFER
TIM A. WHEELER
KYLE H. WRAY

Algorithms for Decision Making
MIT Press, 2022

ALGORITHMS FOR
VALIDATION



MYKEL J. KOCHENDERFER
SYDNEY M. KATZ
ANTHONY L. CORSO
ROBERT J. MOSS

Algorithms for Validation
MIT Press, 2025

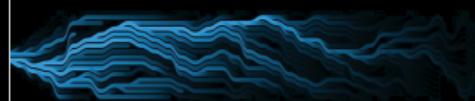
TEXTBOOKS USING JULIA

ALGORITHMS FOR
OPTIMIZATION



MYKEL J. KOCHENDERFER
TIM A. WHEELER

ALGORITHMS FOR
DECISION MAKING



MYKEL J. KOCHENDERFER
TIM A. WHEELER
KYLE H. WRAY

ALGORITHMS FOR
VALIDATION



MYKEL J. KOCHENDERFER
SYDNEY M. KATZ
ANTHONY L. CORSO
ROBERT J. MOSS

Algorithms for Optimization
MIT Press, 2019

Algorithms for Decision Making
MIT Press, 2022

Algorithms for Validation
MIT Press, 2025

All PDFs available for free at <https://algorithmsbook.com>

JULIACON 2019



The slide features a white background with a purple footer bar at the bottom. In the center, there is a video frame showing a presentation. The video frame has a white header bar with the JuliaCon logo and text, and a white footer bar with the speaker's name and affiliation.

HOW WE WROTE A TEXTBOOK USING JULIA
CODE, CONTENT, AND TOOLING

TIM WHEELER
JULY 2019

Tim Wheeler
Kitty Hawk

How We Wrote a Textbook using Julia
Tim Wheeler, JuliaCon 2019

TEXTBOOKS WITH INTEGRATED JULIA

TEXTBOOKS WITH INTEGRATED JULIA

§8 CHAPTER 4: FALSIFICATION THROUGH OPTIMIZATION

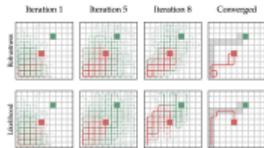


Figure 4.3. A comparison of optimization-based falsification on the grid world using the robustness objective and the weighted objective. The plots show the progression of the robustness-based optimization algorithm, which will be discussed in the next section. The first column of plots in the final column represents the robustness objective, while the second column represents the weighted objective (this follows that quickly moves towards the objective). The weighted objective finds a failure that stops the search early, while the robustness objective only converges toward the solution at the end.

issues, other objective functions may lead to the discovery of more likely failures in practice.

Another common objective for most likely failure analysis is

$$f(\tau) = p(\tau, \eta) - \lambda \log(p(\tau)) \quad (4.8)$$

where λ is a weighting parameter selected by the user (algorithm 4.10). This objective is smooth and encourages the optimization algorithm to search stimulus ready for trajectories that are both likely and close to failure.

```
function weighted_likelihood_objective(x, y, z, w, b, n, l=1.0)
    y = rollnorm(y, v, x)
    c = l * norm(x - y, 1)
    x = [l * ones(1), l * zeros(1), 1]
    p = logpdf(multinomial(y, w, length(w)))
    return softmax(x, w, formula, n) * c + log(pdf(x, c))
end
```

4.6 Optimization Algorithms

We can search for failures by applying a variety of optimization algorithms to the optimization problem in equation (4.5).⁴ Algorithm 4.11 implements

© 2012 Kochenderfer, Kate, Corra, and Mooney under a Creative Commons CC-BY-NC-ND license.
aaron-01-13-12-jarrell, comments to log@jaguar1.us

Algorithm 4.11. Objective function that weights the violation between solutions and likelihood. The function takes a vector x , a weight w , a vector y , a vector v , a scalar c , and a specification b . It returns a weighted combination of the weighted violation for the solutions if $w > 0$ and the weighted likelihood under the normal trajectory distribution.

⁴M. J. Kochenderfer and T.A. Wheeler, *Algorithms for Optimization*. MIT Press, 2020.

Algorithms for Validation Figures and Julia code

TEXTBOOKS WITH INTEGRATED JULIA

58 CHAPTER 4. FALSIFICATION THROUGH OPTIMIZATION

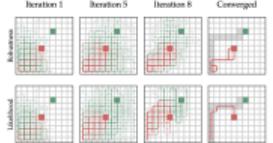


Figure 4.3 shows four 4x4 grid plots illustrating the progress of an optimization-based falsification algorithm. The first three plots are labeled 'Iteration 1', 'Iteration 5', and 'Iteration 8'. The fourth plot is labeled 'Converged'. Each plot contains a red dot representing the current state and green dots representing obstacles. The algorithm starts at the bottom-left corner and moves towards the top-right corner, avoiding obstacles. The 'Converged' plot shows the final state where the algorithm has reached the goal.

Figure 4.3. A comparison of optimization-based falsification on the grid world using the robustness objective and the weighted objective. The plots above show the progression of the importance sampling algorithm, which will be discussed in the next section. The initial state is shown in the first column; the final state in the third column represents the robustness objective, while the final state in the fourth column represents the weighted objective (this finds failures that quickly move towards the obstacles). The last plot shows the failure that stops the algorithm from moving, after many visits toward the obstacles at the end.

issues, other objective functions may lead to the discovery of more likely failures in practice.

Another common objective for most likely failure analysis is

$$f(\tau) = p(\tau, \psi) - \lambda \log(p(\tau)) \quad (4.8)$$

where λ is a weighting parameter selected by the user (algorithm 4.10). This objective is smooth and converges the optimization algorithm to search stimulus ready for trajectories that are both likely and close to failure.

```
function weighted_likelihood_objective(x, y, w, b, smoothness=0.0, N=2-1.0)
    y = rollout(sys, x, y)
    c = rollout(sys, y, x)
    s = [true, i for i in 1:N-1]
    p = [pdf(w, x, formula, s) for _ in 1:N]
    qs = [pdf(b, y, formula, s) for _ in 1:N]
    return softmax(s, w.*formula, w.*smoothness) - b.*log(pdf(x, c))
end
```

4.6 Optimization Algorithms

We can search for failures by applying a variety of optimization algorithms to the optimization problem in equation (4.5).⁴ Algorithm 4.11 implements

© 2013 Kochenderfer, Kate, Corra, and Mooney under a Creative Commons CC-BY-NC-ND license.
algorithm-4.11.jl (github.com/kochenderfer/algosforoptimization), MIT Press, www.

Algorithms for Validation Figures and Julia code

```
struct ImportanceSamplingEstimation
    p # nominal distribution
    q # proposal distribution
    m # number of samples
end

function estimate(alg::ImportanceSamplingEstimation, sys, ψ)
    p, q, m = alg.p, alg.q, alg.m
    ts = [rollout(sys, q) for i in 1:m]
    ps = [pdf(p, τ) for τ in ts]
    qs = [pdf(q, τ) for τ in ts]
    ws = ps ./ qs
    return mean(w * isfailure(ψ, τ) for (w, τ) in zip(ws, ts))
end
```

Example algorithm

Importance sampling estimation of failure probability

JULIA INTEGRATED INTO L^AT_EX CODE

```
368 where the weights are  $w_{i,l} = p(x_{t+1}) / q(x_{t+1})$ . These weights are sometimes referred to as importance weights. Trajectories that are more likely under the nominal trajectory distribution have  
higher importance weights.  
369  
370 %begin{algorithm} % importance sampling estimation  
371 %begin{juliaverbatim}  
372 struct ImportanceSamplingEstimation  
373     p # nominal distribution  
374     q # proposal distribution  
375     n # number of samples  
376 end  
377  
378 function estimate(p,q;isImportanceSamplingEstimation,sys,v)  
379     n = v.alg.n  
380     alg = v.alg  
381     rs = [rollout(sys, q) for i in 1:n]  
382     ps = [pdf(p, r) for r in rs]  
383     qs = [pdf(q, r) for r in rs]  
384     ws = ps ./ qs  
385     return mean(ws * isFailure(rs, v) for (w, r) in zip(ws, rs))  
386 end  
387 %end{juliaverbatim}  
388 %caption{\texttt{isFailure}} The importance sampling estimation algorithm for estimating the  
probability of failure. The algorithm generates  $\mathcal{N}(v_m)$  samples from the proposal distribution  $q(v|q)$ . It then computes the importance weights for the samples and applies \texttt{isFailure} to compute  
whether \texttt{isFailure} is  
%end(algorithm)  
389  
390 %subsection{Optimal Proposal Distribution}  
391 %label{SectionOptimal_Proposal}  
392 The accuracy and efficiency of importance sampling approaches is highly dependent on the proposal  
distribution. The variance of the estimator in \texttt{isFailure} is  
393 %begin{equation}  
394 \text{Var}(\text{isFailure}) = \frac{1}{n} \sum_{i=1}^n \text{Var}(w_i) + \frac{1}{n^2} \sum_{i=1}^n \text{Cov}(w_i, w_j)  
395 %end{equation}  
396 %caption{\texttt{isFailure}}  
397 %label{isFailure_var}  
398 %begin{equation}  
399 \text{Var}(w_i) = \frac{1}{n} \sum_{j=1}^n \text{Var}(p(x_{t+1}|x_t)) = \frac{1}{n} \sum_{j=1}^n \text{Var}(p(x_{t+1}))  
400 %end{equation}  
401 %caption{\texttt{isFailure}}  
402 %label{isFailure}  
403 %begin{equation}  
404 \text{Cov}(w_i, w_j) = \frac{1}{n} \sum_{k=1}^n \text{Cov}(p(x_{t+1}|x_k), p(x_{t+1}))  
405 %end{equation}  
406 This distribution corresponds to the failure distribution  $p(f|u_m \text{ and } \text{true\_within\_psi})$ . As noted in  
%ref{failure_distribution}, computing this distribution is not possible in practice since we often  
do not know the full set of failure trajectories and the normalizing constant  $\int p(f|u_m \text{ and } \text{true\_within\_psi}) df$  is the  
quantity we are trying to estimate. Our goal is therefore to select a proposal distribution that is as  
close as possible to the failure distribution.
```

L^AT_EX code example

Integrated Julia code using [pythontex](#)

JULIA INTEGRATED INTO LATEX CODE

```

368 where the weights are  $w_i = p(x_{t+1}) / q(x_{t+1})$ . These weights are sometimes referred to as xtext (importance weights). Trajectories that are more likely under the nominal trajectory distribution have higher importance weights.
369
370 begin(algorithm) % importance sampling estimation
371   begin(JuliaVerbatim)
372     struct ImportanceSamplingEstimation
373       p # nominal distribution
374       q # proposal distribution
375       n # number of samples
376     end
377
378     function estimate(alg:ImportanceSamplingEstimation, sys, q)
379       n, u = alg.n, alg.u
380       alg = alg(sys, q)
381       rs = rollout(q, t) for t in 1:n
382       ps = pdf(q, r) for r in rs
383       qs = pdf(u, r) for r in rs
384       ws = ps ./ qs
385       ws = ws ./ sum(ws)
386       return measure = isfailure(u, r) for (w, r) in zip(ws, rs)
387     end
388   end(JuliaVerbatim)
389   caption(alg:ImportanceSamplingEstimation) The importance sampling estimation algorithm for estimating the probability of failure. The algorithm generates N(m) samples from the proposal distribution p|q. It then computes the importance weights for the samples and applies xref{eq:is_estimator} to compute xlabel{xtext}(fail).
390   end(algorithm)
391
392   subsection(Optimal Proposal Distribution)
393   label(optimal_proposal)
394   The accuracy and efficiency of importance sampling approaches is highly dependent on the proposal distribution and variance of the estimator in xref{eq:is_estimator} is
395   begin(Equation)
396     label(eq_is_var)
397     \text{Var}[\hat{p}_{\text{fail}}] = \frac{1}{N} \sum_{i=1}^N \left[ \frac{p(x_i) \mathbb{E}[x_i | \text{fail}] - \hat{p}_{\text{fail}} p(x_i)}{q(x_i)} \right]^2
398   end(Equation)
399   In general we want to select a proposal distribution that makes this variance low, and the optimal proposal distribution is the one that minimizes this variance.
400
401 It is evident from xref{eq:is_var} that we can achieve a variance of zero when
402 begin(Equation)
403   label(eq_optimal_proposal)
404   
$$q(x) = p(x) / \mathbb{E}[x | \text{fail}]$$

405   end(Equation)
406 This distribution corresponds to the failure distribution np(xfail | xnom, pfail). As noted in xref{failure_distribution}, computing this distribution is not possible in practice since we often do not know the full set of failure trajectories and the normalizing constant pfail is the quantity we are trying to estimate. Our goal is therefore to select a proposal distribution that is as close as possible to the failure distribution.

```

LATEX code example
Integrated Julia code using `pythontex`

7.2. IMPORTANCE SAMPLING 147

where the weights are $w_i = p(x_i)/q(x_i)$. These weights are sometimes referred to as `xtext` (importance weights). Trajectories that are more likely under the nominal trajectory distribution have higher importance weights.

```

struct ImportanceSamplingEstimation
  p # nominal distribution
  q # proposal distribution
  n # number of samples
end

function estimate(alg:ImportanceSamplingEstimation, sys, q)
  n, u = alg.n, alg.u, alg.a
  rs = rollout(q, t) for t in 1:n
  ps = pdf(q, r) for r in rs
  qs = pdf(u, r) for r in rs
  ws = ps ./ qs
  ws = ws ./ sum(ws)
  return measure = isfailure(u, r) for (w, r) in zip(ws, rs)
end

```

Algorithm 7.3. The importance sampling estimation algorithm for estimating the probability of failure. The algorithm generates `N(m)` samples from the proposal distribution. It then computes the importance weights for the samples and applies equation (7.9) to compute `fail`.

7.2.2 Optimal Proposal Distribution

The accuracy and efficiency of importance sampling approaches is highly dependent on the proposal distribution. The variance of the estimator in equation (7.8) is

$$\text{Var}[\hat{p}_{\text{fail}}] = \frac{1}{N} \mathbb{E}_{i \sim q(\cdot)} \left[\frac{(p(x_i) \mathbb{E}[x_i | \text{fail}] - \hat{p}_{\text{fail}} p(x_i))^2}{q(x_i)} \right] \quad (7.10)$$

In general, we want to select a proposal distribution that makes this variance low, and the optimal proposal distribution is the one that minimizes this variance.

It is evident from equation (7.9) that we can achieve a variance of zero when

$$q^*(x) = \frac{p(x) \mathbb{E}[x | \text{fail}]}{p_{\text{fail}}} \quad (7.11)$$

This distribution corresponds to the failure distribution $p(x | \text{fail}) / p_{\text{fail}}$. As noted in chapter 6, computing this distribution is not possible in practice since we often do not know the full set of failure trajectories and the normalizing constant p_{fail} is the quantity we are trying to estimate. Our goal is therefore to select a proposal distribution that is as close as possible to the failure distribution.

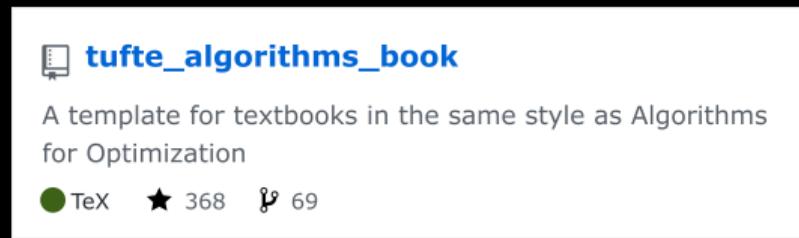
© 2024 Koehnlein, Katz, Cane, and Mous shared under a Creative Commons CC-BY-NC-ND license.

2024-01-13 13:31:08, comments to koehnlein@mit.edu

Compiled LATEX PDF
Julia algorithms and LATEX math

OPEN-SOURCE TEXTBOOK TEMPLATE REPOSITORY

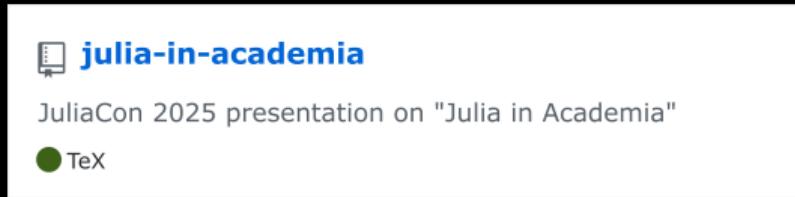
OPEN-SOURCE TEXTBOOK TEMPLATE REPOSITORY



github.com/sisl/tufte_algorithms_book

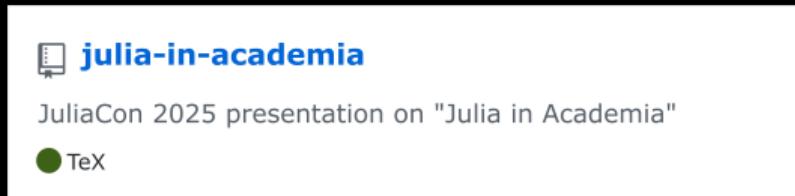
OPEN-SOURCE SLIDES REPO (`BEAMER` + `PYTHONTEX`)

OPEN-SOURCE SLIDES REPO (BEAMER + PYTHONTEX)

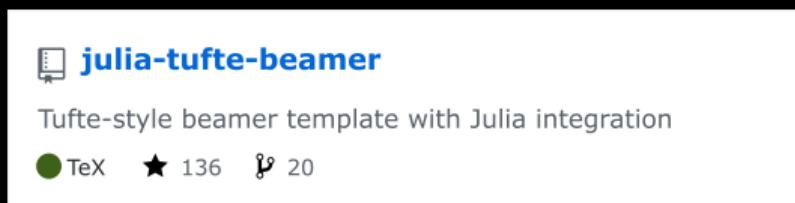


github.com/mossr/julia-in-academia

OPEN-SOURCE SLIDES REPO (BEAMER + PYTHONTEX)



github.com/mossr/julia-in-academia



github.com/mossr/julia-tufte-beamer

INTEGRATED JULIA EXAMPLE

We can compute Julia code *directly in the slides/pages.*

INTEGRATED JULIA EXAMPLE

We can compute Julia code *directly in the slides/pages.*

```
julia> using SignalTemporalLogic
```

INTEGRATED JULIA EXAMPLE

We can compute Julia code *directly in the slides/pages*.

```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, 2.0, 1.5, 3.0, 0.5, -0.5, -2.0, -4.0, -1.5];
julia> ψ₁ = @formula ◊(sₜ → sₜ > 0);

τ # \tau<TAB>
ψ₁ # \psi<TAB>\_1<TAB>
◊ # \lozenge<TAB>
```

INTEGRATED JULIA EXAMPLE

We can compute Julia code *directly in the slides/pages*.

```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, 2.0, 1.5, 3.0, 0.5, -0.5, -2.0, -4.0, -1.5];
julia> ψ₁ = @formula ◊(sₜ → sₜ > 0);
julia> ρ₁ = ρ(τ, ψ₁)
3.0
```

```
τ # \tau<TAB>
ψ₁ # \psi<TAB>\_1<TAB>
◊ # \lozenge<TAB>
ρ # \rho<TAB>
```

INTEGRATED JULIA EXAMPLE

We can compute Julia code *directly in the slides/pages*.

```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, 2.0, 1.5, 3.0, 0.5, -0.5, -2.0, -4.0, -1.5];
julia> ψ₁ = @formula ◊(sₜ → sₜ > 0);
julia> ρ₁ = ρ(τ, ψ₁)
3.0
julia> ψ₂ = @formula □(sₜ → sₜ > 0);
```

```
τ # \tau<TAB>
ψ₁ # \psi<TAB>\_1<TAB>
◊ # \lozenge<TAB>
ρ # \rho<TAB>
□ # \square<TAB>
```

INTEGRATED JULIA EXAMPLE

We can compute Julia code *directly in the slides/pages*.

```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, 2.0, 1.5, 3.0, 0.5, -0.5, -2.0, -4.0, -1.5];
julia> ψ₁ = @formula ◊(sₜ → sₜ > 0);
julia> ρ₁ = ρ(τ, ψ₁)
3.0
julia> ψ₂ = @formula □(sₜ → sₜ > 0);
julia> ρ₂ = ρ(τ, ψ₂)
-4.0

τ # \tau<TAB>
ψ₁ # \psi<TAB>\_1<TAB>
◊ # \lozenge<TAB>
ρ # \rho<TAB>
□ # \square<TAB>
```

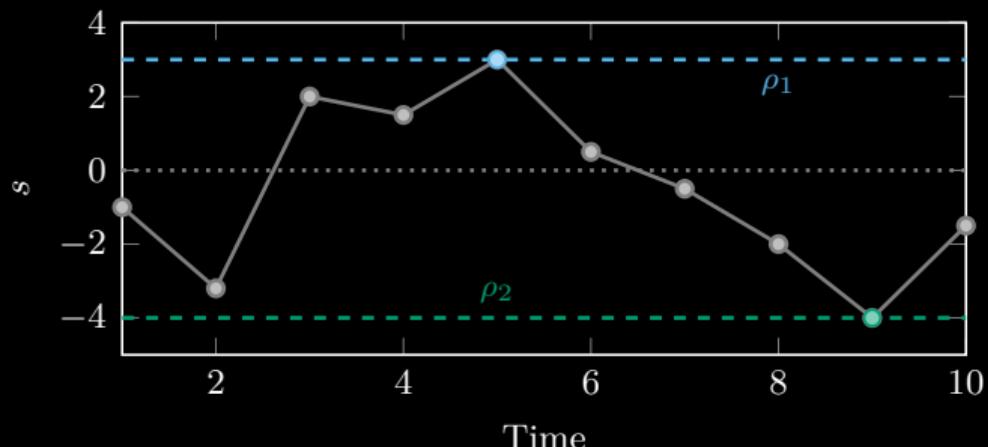
INTEGRATED JULIA EXAMPLE

We can compute Julia code *directly in the slides/pages*.

```
julia> using SignalTemporalLogic
julia> τ = [-1.0, -3.2, 2.0, 1.5, 3.0, 0.5, -0.5, -2.0, -4.0, -1.5];
julia> ψ₁ = @formula ◊(sₜ → sₜ > 0);
julia> ρ₁ = ρ(τ, ψ₁)
3.0
julia> ψ₂ = @formula □(sₜ → sₜ > 0);
julia> ρ₂ = ρ(τ, ψ₂)
-4.0

τ # \tau<TAB>
ψ₁ # \psi<TAB>\_1<TAB>
◊ # \lozenge<TAB>
ρ # \rho<TAB>
□ # \square<TAB>
```

$$\psi_1 = \diamond(s_t > 0) \quad \psi_2 = \square(s_t > 0)$$



JULIA FOR TEXTBOOKS: THE GOOD

JULIA FOR TEXTBOOKS: THE GOOD

- Concise algorithm descriptions (fits within single page)

JULIA FOR TEXTBOOKS: THE GOOD

- Concise algorithm descriptions (fits within single page)
- Multiple dispatch makes it easy to design common interface

JULIA FOR TEXTBOOKS: THE GOOD

- Concise algorithm descriptions (fits within single page)
- Multiple dispatch makes it easy to design common interface
- Auto-differentiation allows for clean algorithms that just work

JULIA FOR TEXTBOOKS: THE GOOD

- Concise algorithm descriptions (fits within single page)
- Multiple dispatch makes it easy to design common interface
- Auto-differentiation allows for clean algorithms that just work
- Integrated tooling to run and plot Julia code during PDF compilation

JULIA FOR TEXTBOOKS: THE GOOD

- Concise algorithm descriptions (fits within single page)
- Multiple dispatch makes it easy to design common interface
- Auto-differentiation allows for clean algorithms that just work
- Integrated tooling to run and plot Julia code during PDF compilation
- Full Unicode support means math matches code (e.g., λ in math and `\lambda` in code)

JULIA FOR TEXTBOOKS: THE GOOD

- Concise algorithm descriptions (fits within single page)
- Multiple dispatch makes it easy to design common interface
- Auto-differentiation allows for clean algorithms that just work
- Integrated tooling to run and plot Julia code during PDF compilation
- Full Unicode support means math matches code (e.g., λ in math and `\lambda` in code)
- Seamless package integration to not reinvent the wheel

JULIA FOR TEXTBOOKS: THE GOOD

- Concise algorithm descriptions (fits within single page)
- Multiple dispatch makes it easy to design common interface
- Auto-differentiation allows for clean algorithms that just work
- Integrated tooling to run and plot Julia code during PDF compilation
- Full Unicode support means math matches code (e.g., λ in math and `λ` in code)
- Seamless package integration to not reinvent the wheel

Shout-out to the following packages:

`Distributions.jl`, `LazySets.jl`, `IntervalArithmetic.jl`, `Flux.jl`, `Optim.jl`, and `JuMP.jl`

SHOWCASE: `IntervalArithmetic.jl` and `LazySets.jl`

SHOWCASE: `IntervalArithmetic.jl` and `LazySets.jl`

```
struct NaturalInclusion <: ReachabilityAlgorithm
    h # time horizon
end

function r(sys, x)
    s, x = extract(sys.env, x)
    τ = rollout(sys, s, x)
    return τ[end].s
end

to_hyperrectangle(I) = Hyperrectangle(low=[i.lo for i in I],
                                         high=[i.hi for i in I])

function reachable(alg::NaturalInclusion, sys)
    I's = []
    for d in 1:alg.h
        I = intervals(sys, d)
        push!(I's, r(sys, I))
    end
    return UnionSetArray([to_hyperrectangle(I') for I' in I's])
end
```

SHOWCASE: `IntervalArithmetic.jl` and `LazySets.jl`

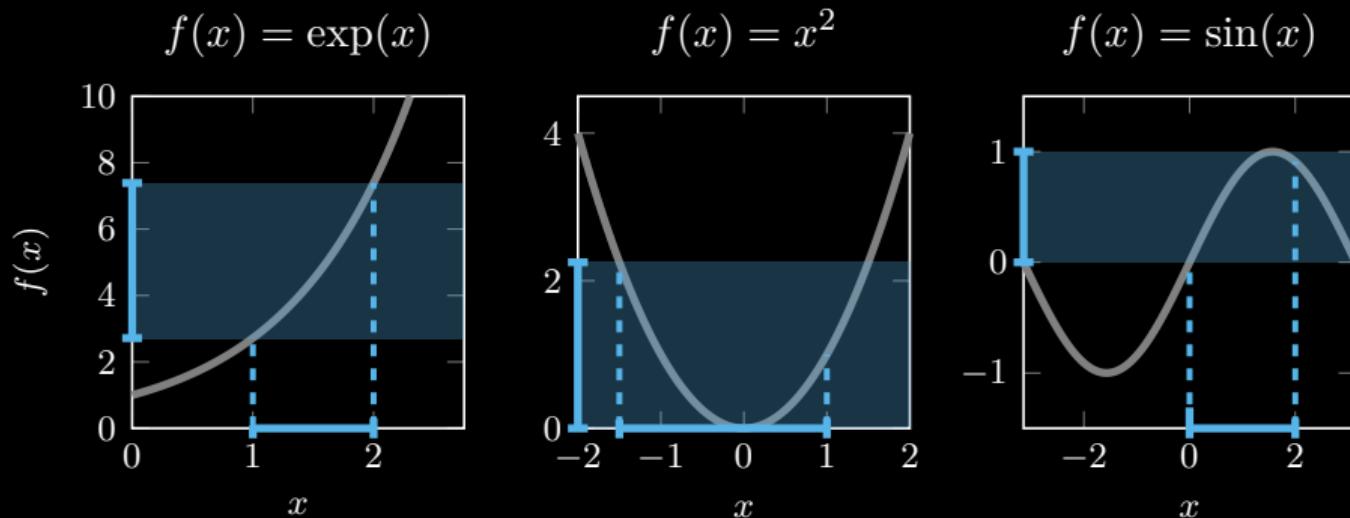


Figure: Example of the interval counterparts for the \exp , square, and \sin functions.

SHOWCASE: `IntervalArithmetic.jl` and `LazySets.jl`

```
function (env::InvertedPendulum)(s, a)
    θ, ω = s[1], s[2]
    dt, g, m, l = env.dt, env.g, env.m, env.l
    ω = ω + (3g / (2 * l) * sin(θ) + 3 * a / (m * l^2)) * dt
    θ = θ + ω * dt
    return [θ, ω]
end
```

SHOWCASE: `IntervalArithmetic.jl` and `LazySets.jl`

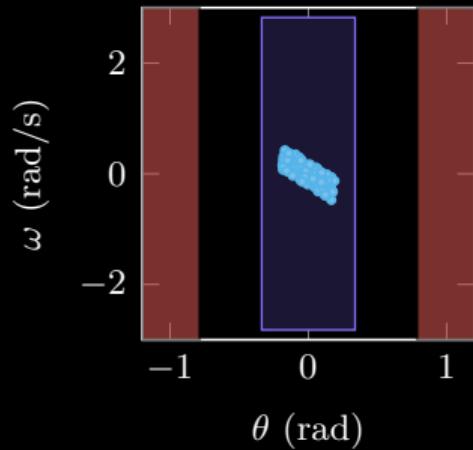


Figure: Reachable set for the inverted pendulum system.

JULIA FOR TEXTBOOKS: THE NOT-SO-GOOD

JULIA FOR TEXTBOOKS: THE NOT-SO-GOOD

- Somewhat new to readers: Requires learning Julia in the process

JULIA FOR TEXTBOOKS: THE NOT-SO-GOOD

- Somewhat new to readers: Requires learning Julia in the process
- Potential obscure idioms: broadcasting, anonymous functions, multiple dispatch

JULIA FOR TEXTBOOKS: THE NOT-SO-GOOD

- Somewhat new to readers: Requires learning Julia in the process
- Potential obscure idioms: broadcasting, anonymous functions, multiple dispatch
- Textbook tooling: Needed to be built out from scratch

JULIA IN THE CLASSROOM

JULIA IN THE CLASSROOM

Stanford University

AA228V/CS238V

Validation of Safety Critical Systems

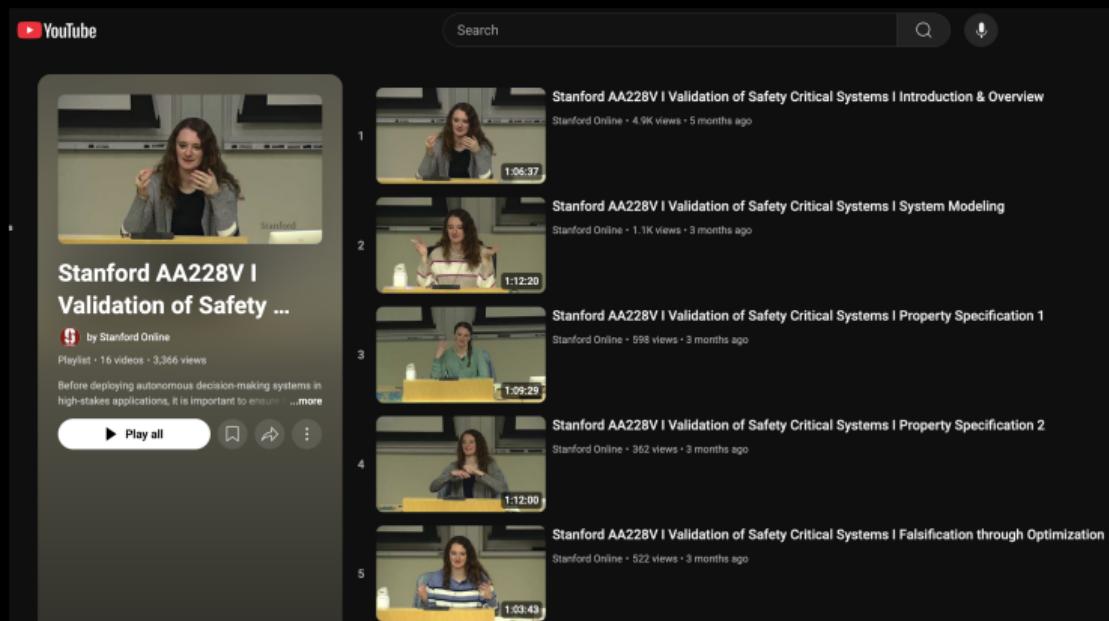
Grad-level course at Stanford

Follows *Algorithms for Validation* textbook

First offered in Winter 2025

LECTURES

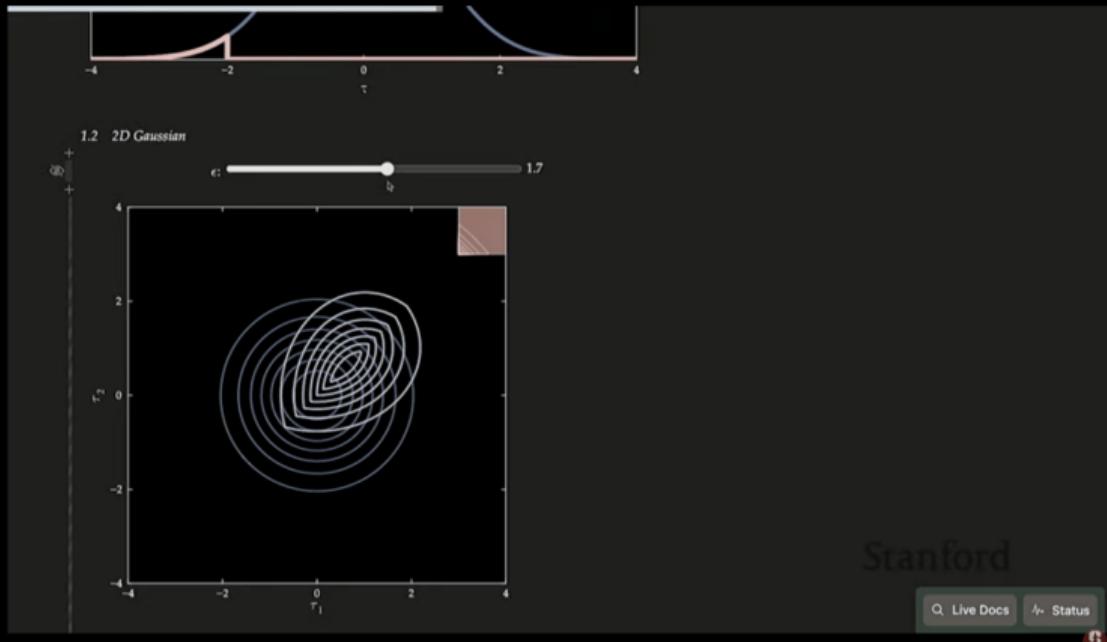
LECTURES



All lectures available on YouTube

Led by Sydney Katz, textbook co-author and award-winning lecturer

INTERACTIVE LECTURES



Interactive lectures using [Pluto.jl](#)

ASSIGNMENTS IN JULIA

ASSIGNMENTS IN JULIA

Stanford AA228V/CS238V Programming Projects

Programming projects for Stanford's AA228V/CS238V Validation of Safety-Critical Systems.

CAS: Failure

CAS: Success

Uses the [StanfordMAster_11](#) Julia package.

Projects

- Project 0 — A light-weight introduction to telelocation.
- Project 1 — Efficiently finding the near-shore failure.
- Project 2 — Efficiently estimating the failure probability.
- Project 3 — Estimating the volume of a reachable set.

Assignments repository
Student-facing code

ASSIGNMENTS IN JULIA

Stanford AA228V/CS238V Programming Projects

Programming projects for Stanford's AA228V/CS238V Validation of Safety-Critical Systems.

CAS: Failure CAS: Success

Uses the [StanfordMATH.jl](#) Julia package.

Projects

- Project 0 – A light-weight introduction to Gradscale.
- Project 1 – Efficiently finding the near-shley failure.
- Project 2 – Efficiently estimating the failure probability.
- Project 3 – Estimating the volume of a macheable set.

Assignments repository
Student-facing code

Project 2: Estimating failure probability

Please wait until the entire notebook is finished loading before proceeding (you may get temporary errors).

Task: Efficiently estimating the failure probability using n total calls to the system step function.

- Small system: 1D Gaussian $\mathcal{N}(0, 1)$; $n = 200$ step calls and 20 seeds.
- Medium system: Swinging inverted pendulum; $n = 10,000$ step calls and 5 seeds.
- Large system: Aircraft collision avoidance system (CAS); $n = 200,000$ step calls and 3 seeds.

(Same system as Project 1)

Your job is to write the following function that returns the estimated failure probability:

```
estimate_probability(ys, ys; n)::Float64
```

and get a better estimate of the failure probability than a random baseline.

See the three "Task" sections below for where to fill out the algorithms.

Submission

You will submit three results files (.val) and this file (project2.jl) to the "Project 2 (.val files + .jl file)" GradeScope assignment and include the PDF in your write up in the "Project 2 (write up)" GradeScope assignment (see below).

Gradscale assignment "Project 2 (.val files + .jl file)" (total 6 points)

- project2.jl (0 points)
- project2-small.val (0 points)
- project2-medium.val (2 points)
- project2-large.val (3 points)

Table of Contents

- Project 2: Estimating failure probability ..
- Submission ..
- Algorithm write up ..
- Export to PDF ..
- TestPlan ..
- JuliaPkgInfo ..
- Hints ..
- Common errors ..

Small: 1D Gaussian

- Small system ..
- Small environment ..
- Small specification ψ ..
- Probability vs. likelihood ..
- Probability of an event ..
- Baseline: Monte Carlo estimate ..
- Baseline comparison ..
- Baseline results (small) ..
- Average performance ..
- Example aggregate performance fo..
- ... Task (Small): Estimate failure prob..

Small Tests

- Different failure threshold ..
- Random failure threshold ..
- Created small test (0? points)

Medium: Inverted Pendulum

- Medium system ..
- Medium specification ψ ..
- Baseline: Medium ..
- Average performance ..
- Example aggregate performance fo..
- ... Task (Medium): Estimate failure ..

Medium Test

Pluto assignments
Includes local tests

ASSIGNMENTS IN JULIA

The screenshot shows a Pluto notebook interface with the following details:

- Title:** Large Test
- Description:** We'll automatically test your most_likely_failure(::largeSystem, s) function below.
- Status:** ✓ Graded large test (3/3 points)
- Feedback:** Click to re-run the LargeSystem evaluation.
- Code Output:** This will re-run most_likely_failure(::LargeSystem, s) and re-save project3-large.vat. Uncheck this to load results from the file.
- Plot:** CollisionAvoidance tests passed! A plot titled "Most-likely failure found" shows position A (m) versus time t_end (s). The plot shows a red curve starting at approximately (-100, -100), rising to a peak around (30, 100), and then falling back towards the end point.
- Text Output:** You found a passing trajectory!
 $\ell = -97.37071722814194$ (failure log-likelihood)
 $n_{step} = 9.963$ (step calls ≤ 10,000)
- Table of Contents:**
 - Project 1: Finding the most-likely fail...
 - Submission
 - Algorithm write up
 - Export to PDF
 - Textbook
 - JuliaPluto tips
 - Hints
 - Common errors
- Small Tests:**
 - Small: 1D Gaussian
 - Small system
 - Small environment
 - Small specification φ
 - Random baseline φ
 - Baselines
 - Baseline results (small)
 - Task (Small): Most-likely failure
 - Small Answers
 - Fuzzing
 - Fuzzing Implementation
 - Optimization
 - Optim.jl
 - Optim.JL Nelder-Mead Implement...
 - (Custom) Gradient Descent
 - (Custom) Gradient Descent Implement...
 - Small Tests
 - Deadline checker threshold
 - Random failure threshold
 - Slider to control threshold
 - Graded small test (1/1 points)
 - Algorithm for SmallSystem
- Bottom Navigation:** Live Docs, Status

Interactive Pluto tests
Get feedback instantly

ASSIGNMENTS IN JULIA

The screenshot shows a Pluto notebook interface with the following details:

- Title:** Large Test
- Description:** We'll automatically test your most_likely_failure(::largeSystem, s) function below.
- Status:** Graded large test (3/3 points)
- Feedback:** Click to re-run the LargeSystem evaluation.
- Notes:** This will re-run most_likely_failure(::LargeSystem, s) and re-save project3-large.vat. Uncheck this to load results from the file.
- Plot:** CollisionAvoidance tests passed! A plot titled "Most-likely failure found" showing position A (m) vs time t_end (s). The plot shows a red curve starting at approximately (-100, -100), rising to a peak around (30, 100), and then falling back towards the end point.
- Results:** Results saved for CollisionAvoidance
- File:** /Users/moser/Code/sis1/AA228V/Student/AA228VProjects/project3/project3-large.vat
- Instructions:** Please submit the file listed above to Gradescope.

- Separated assignment code from core library ([StanfordAA228V.jl](#))

Interactive Pluto tests
Get feedback instantly

ASSIGNMENTS IN JULIA

The screenshot shows a Pluto notebook interface with the following details:

- Title:** Large Test
- Description:** Will automatically test your most_likely_failure(::largeSystem, s) function below.
- Status:** Graded large test (3/3 points)
- Feedback:** Click to re-run the LargeSystem evaluation.
- Plot:** CollisionAvoidance tests passed! A plot showing A (m) on the y-axis (ranging from -300 to 300) versus t_end (s) on the x-axis (ranging from 0 to 40). The plot shows a red curve labeled "Most-likely failure found" that starts at approximately (-10, 0), rises to a peak around (30, 100), and ends at approximately (40, -10).
- Text:** You found a passing trajectory!
 $\ell = -97.37071722814194$ (failure log-likelihood)
 $n_{step} = 9.963$ (step calls ≤ 10,000)
- Table of Contents:**
 - Project 1: Finding the most-likely fail...
 - Submission
 - Algorithm write up
 - Export to PDF
 - Textbook
 - JuliaPluto tips
 - Hints
 - Common errors
- Small Tests:**
 - Small: 1D Gaussian
 - Small system
 - Small environment
 - Small specification φ
 - Random baseline
 - Baseline: Small
 - Baseline: Results (small)
 - Task (Small): Most-likely failure
 - Small Answers
- Fuzzing:**
 - Fuzzing Implementation
 - Optimization
 - Optim.jl
 - Optim.jl, Nelder-Mead Implement...
 - (Custom) Gradient Descent
 - (Custom) Gradient Descent Imple...
- Large Tests:**
 - Large: Collision Avoidance
 - Large: Collision Avoidance - Threshold
 - Large: Collision Avoidance - Sliding Window
 - Large: Collision Avoidance - Sliding Window - Threshold
 - Graded small test (1/1 points)
 - Algorithm for SmallSystem
- Bottom:** Please submit the file listed above to Gradescope.

- Separated assignment code from core library ([StanfordAA228V.jl](#))
- Local tests exactly match graded tests

Interactive Pluto tests
Get feedback instantly

ASSIGNMENTS IN JULIA

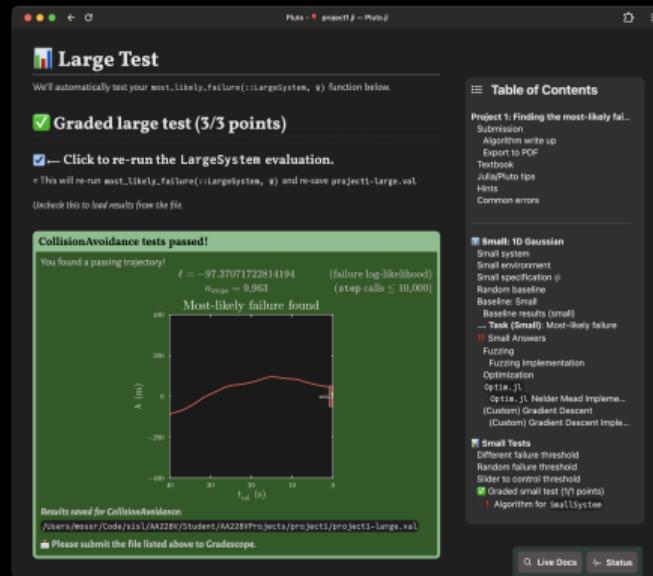
The screenshot shows a Pluto notebook interface with the following details:

- Title:** Large Test
- Description:** Will automatically test your most_likely_failure(::largeSystem, s) function below.
- Status:** Graded large test (3/3 points)
- Feedback:** Click to re-run the LargeSystem evaluation.
- Plot:** CollisionAvoidance tests passed! A plot titled "Most-likely failure found" shows position A (m) vs time t_end (s). The plot shows a red curve starting at approximately (-100, -100), rising to a peak around (30, 100), and then settling near (50, -50).
- Results:** Results saved for CollisionAvoidance.
- File Path:** /Users/moser/Code/sis1/AA228V/Student/AA228VProjects/projects/project3/project3-Large.val
- Instructions:** Please submit the file listed above to Gradescope.

- Separated assignment code from core library ([StanfordAA228V.jl](#))
- Local tests exactly match graded tests
- Interactive nature allows students to play with their algorithms

Interactive Pluto tests
Get feedback instantly

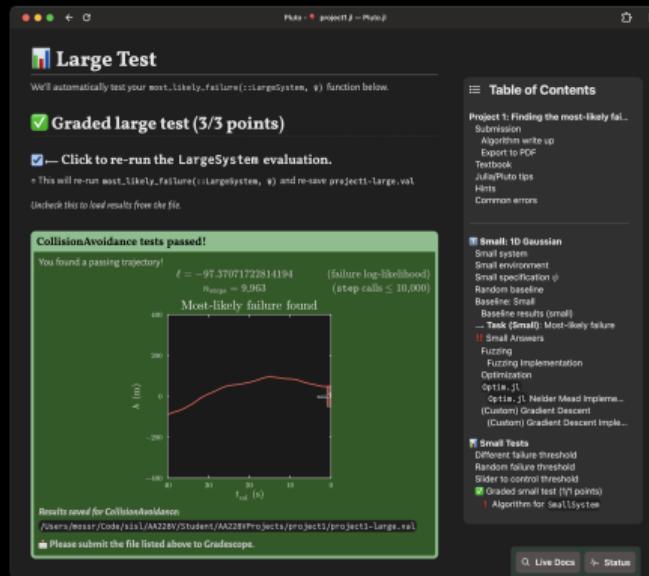
ASSIGNMENTS IN JULIA



- Separated assignment code from core library ([StanfordAA228V.jl](#))
- Local tests exactly match graded tests
- Interactive nature allows students to play with their algorithms
- No “hidden state” that may confuse students

Interactive Pluto tests
Get feedback instantly

ASSIGNMENTS IN JULIA



Interactive Pluto tests
Get feedback instantly

- Separated assignment code from core library ([StanfordAA228V.jl](#))
- Local tests exactly match graded tests
- Interactive nature allows students to play with their algorithms
- No “hidden state” that may confuse students
- Open-source nature requires clever obfuscation of solution code

GRADING ASSIGNMENTS

GRADING ASSIGNMENTS

The screenshot shows the 'Configure Autograder' page on Gradescope. At the top, there's a navigation bar with three dots, a back arrow, and a forward arrow. The title 'Configure Autograder | Gradescope' is displayed, along with the Gradescope logo.

The main content area has a heading 'Configure Autograder'. Below it, a note says: 'Upload your autograder code and change settings here. You can also come back to this step later, but submissions will not be automatically graded until then. Please follow our [guidelines](#) for structuring your autograder.' A note below that says: 'Note: Uploading an autograder zip file will automatically update your Dockerhub image name once it is built successfully.'

A section titled 'Autograder Configuration' contains a note: 'Required field'. It includes a radio button for 'Zip file upload' (which is selected) and another for 'Manual Docker configuration'. Below this is a file input field containing 'autograder_project1.zip', with buttons to 'Replace Autograder (.zip)' and 'Download Autograder'.

Below the file input are dropdown menus for 'Base Image OS' (Ubuntu), 'Base Image Version' (22.04), and 'Base Image Variant' (Base). A note next to these dropdowns says: 'Choose the [base image](#) that will be used to build your autograder. This determines the operating system version and packages available in your autograder.'

At the bottom of this section are two buttons: 'Update Autograder' and 'Test Autograder' (with a play icon).

The 'Docker Image Status' section shows: 'built as of Jan 31, 2025 at 8:58:58 PM PST'. It has sections for 'Build Output' and 'Build Errors'.

At the bottom right of the main content area is a green button labeled 'Manage Submissions'.

Autograde via Gradescope
Students upload Pluto notebook

GRADING ASSIGNMENTS

The screenshot shows the 'Configure Autograder | Gradescope' page. It includes fields for 'Autograder Configuration' (with 'Zip file upload' checked), 'Autograder (.zip)' (containing 'autograder_project1.zip'), and 'Docker Image Status' (built on Jan 31, 2025). A 'Manage Submissions' button is at the bottom.

Autograde via Gradescope
Students upload Pluto notebook

The screenshot shows the GitHub repository 'Gradescope.jl'. It displays the repository's README, which includes a link to the Gradescope autograder specification. The repository has 7 stars, 2 forks, and 7 watching. It also lists releases, packages, and languages (Julia 100%).

Light-weight **Gradescope.jl** package
Manage deps and create autograders

FUTURE USE OF JULIA IN ACADEMIA

FUTURE USE OF JULIA IN ACADEMIA

- Teach fundamentals like probability and statistics in pure Julia

FUTURE USE OF JULIA IN ACADEMIA

- Teach fundamentals like probability and statistics in pure Julia
- Unify curriculum across compounding courses

FUTURE USE OF JULIA IN ACADEMIA

- Teach fundamentals like probability and statistics in pure Julia
- Unify curriculum across compounding courses
- Build cohesive teaching materials using Pluto (e.g., through JuliaAcademy)

FUTURE USE OF JULIA IN ACADEMIA

- Teach fundamentals like probability and statistics in pure Julia
- Unify curriculum across compounding courses
- Build cohesive teaching materials using Pluto (e.g., through JuliaAcademy)
- Build general grading frameworks to work across courses

FUTURE USE OF JULIA IN ACADEMIA

- Teach fundamentals like probability and statistics in pure Julia
- Unify curriculum across compounding courses
- Build cohesive teaching materials using Pluto (e.g., through JuliaAcademy)
- Build general grading frameworks to work across courses
- Compile Julia code to binaries to avoid obfuscation

FUTURE USE OF JULIA IN ACADEMIA

- Teach fundamentals like probability and statistics in pure Julia
- Unify curriculum across compounding courses
- Build cohesive teaching materials using Pluto (e.g., through JuliaAcademy)
- Build general grading frameworks to work across courses
- Compile Julia code to binaries to avoid obfuscation
- Write interactive research papers in Julia/Pluto

What if we could interact directly with the code in research papers?

The screenshot shows a web-based interface for an interactive research paper. At the top, there's a navigation bar with the title 'Algorithms for Validation' and a back arrow. Below the title, there's a section titled 'Lecture Introduction'.

1 Introduction

Before designing decision-making processes in high-stakes settings, it is important to ensure that they will operate as intended. We refer to the process of analyzing the behavior of these systems as validation. Validation is a critical component of the development process for decision-making systems in a variety of domains including autonomous vehicles, robotics, and healthcare. As systems become more complex, the range of behaviors they can exhibit increases, and the spectrum of possible behaviors becomes more challenging and requires a rigorous validation process. This book discusses these challenges and presents a variety of computational methods for validation that can be used to help build a better understanding of system behavior. We motivate the need for validation from a historical perspective and outline the societal consequences of validation failure. We then introduce the validation framework that we will use throughout the book. We discuss the challenges associated with validation and conclude with an overview of the remaining chapters in the book.

2 Probability Distributions

The univariate normal distribution.¹

$$N(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

¹ Here is a longer sidebar explaining more things in detail. This could reference links to other material.

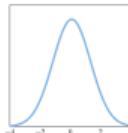


Figure 1: The normal (Gaussian) distribution

$\mu =$ 0.1

$\sigma^2 =$ 1.0

Interactive research papers
Run code directly in the paper

PlutoPapers.jl

Pluto.jl

Algorithms for Validation

Lecture Introduction

1 Introduction

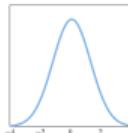
Before designing decision-making processes in high-stakes settings, it is important to ensure that they will operate as intended. We refer to the process of analyzing the behavior of these systems as validation. Validation is a critical component of the development process for decision-making systems in a variety of domains including autonomous vehicles, robotics, and healthcare. As systems become more complex, the range of behaviors they can exhibit increases, and the spectrum of possible behaviors becomes more challenging and requires a rigorous validation process. This book discusses these challenges and presents a variety of computational methods for validation that can be used to help validate a broad range of systems.

We motivate the need for validation from a historical perspective and outline the societal consequences of validation failures. We then introduce the validation framework that we will use throughout the book. We discuss the challenges associated with validation and conclude with an overview of the remaining chapters in the book.

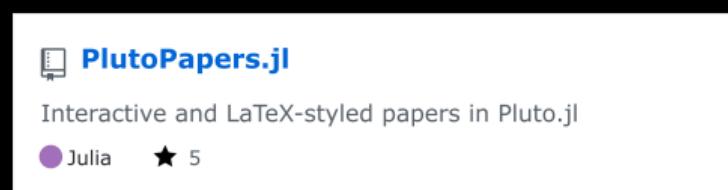
2 Probability Distributions

The univariate normal distribution:

$$N(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma^2} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Figure 1: The normal (Gaussian) distribution

$\mu =$
 $\sigma^2 =$



<https://github.com/mossr/PlutoPapers.jl>

Interactive research papers
Run code directly in the paper

THANK YOU!



QUESTIONS?

mossr@cs.stanford.edu