Hassle-Free Prediction of Excitation Wave Quenching using Actors

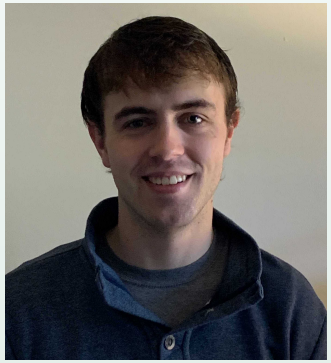
Mahdi Moayeri

Department of Computer Science, University of Saskatchewan

Go20 Conference on Scientific Computing and Software
Marsalforn, Gozo
May 19–23, 2025

Acknowledgements

R. Spiteri



K. Klenk

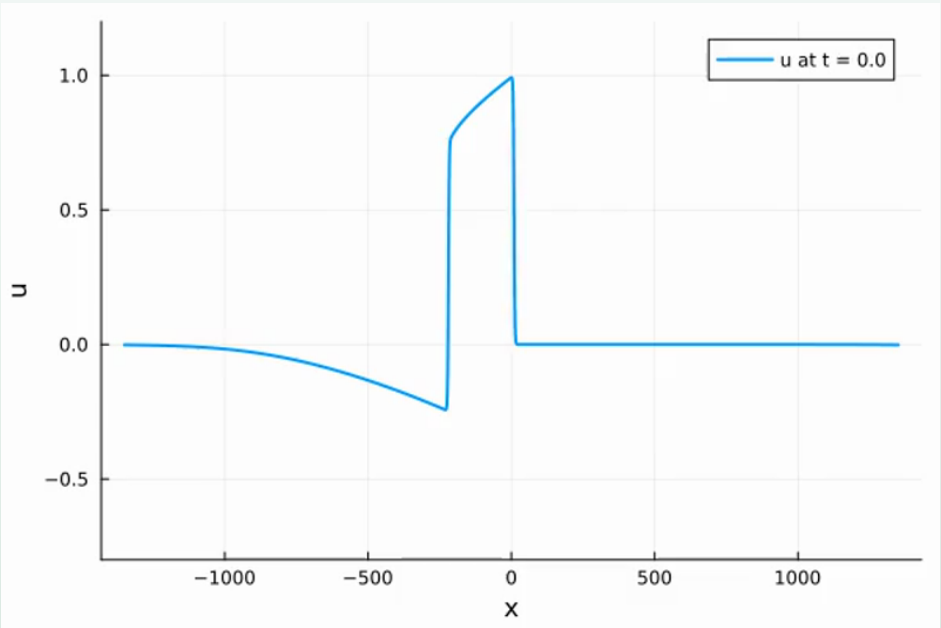
C. Marcotte

NSERC CRSNG

People. Discovery. Innovation.

Outline

Eliminate undesirable excitations

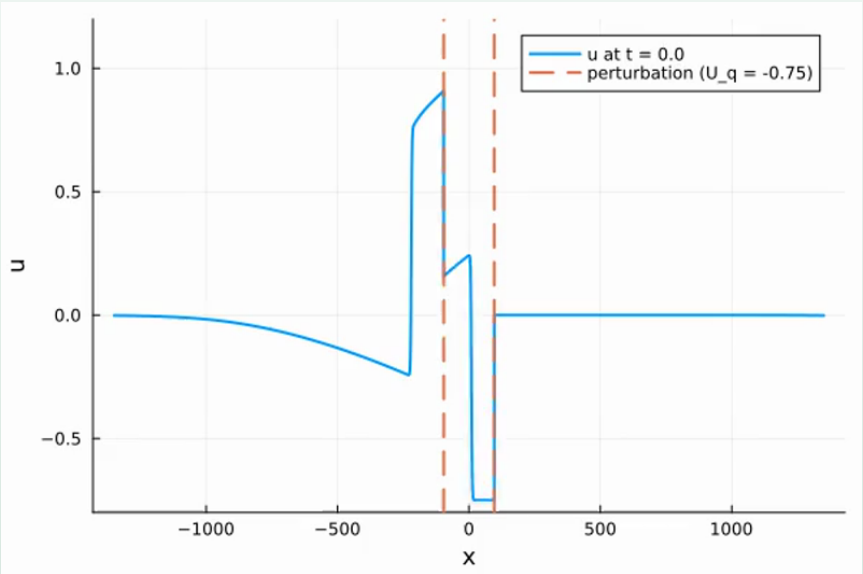


- Excitable media (e.g., nerve/cardiac tissue) support traveling waves
- Stable waves propagate indefinitely, but they can be quenched by perturbations

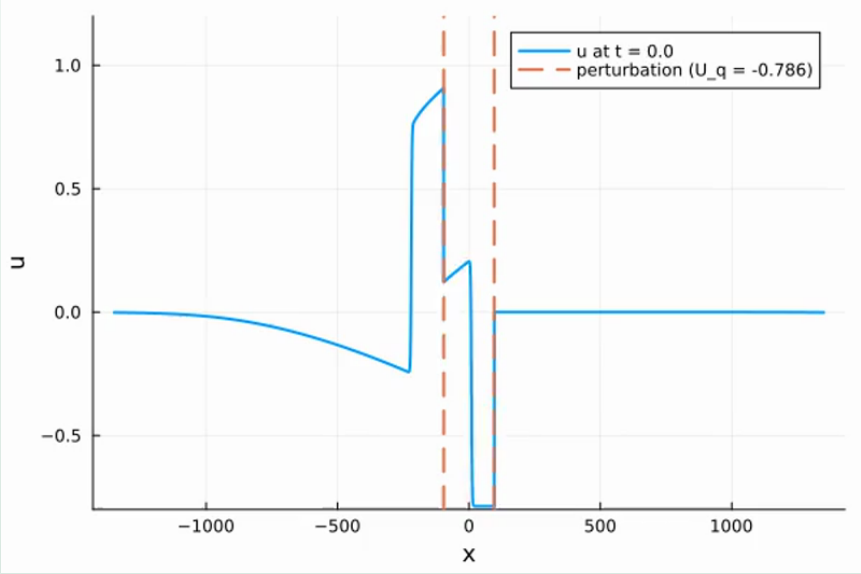
Marcotte, C. D. (2024), Predicting effective quenching of stable pulses in slow-fast excitable media, Physical Review E, 110(6), 064210.

Unquenched vs. Quenched

Unquenched wave

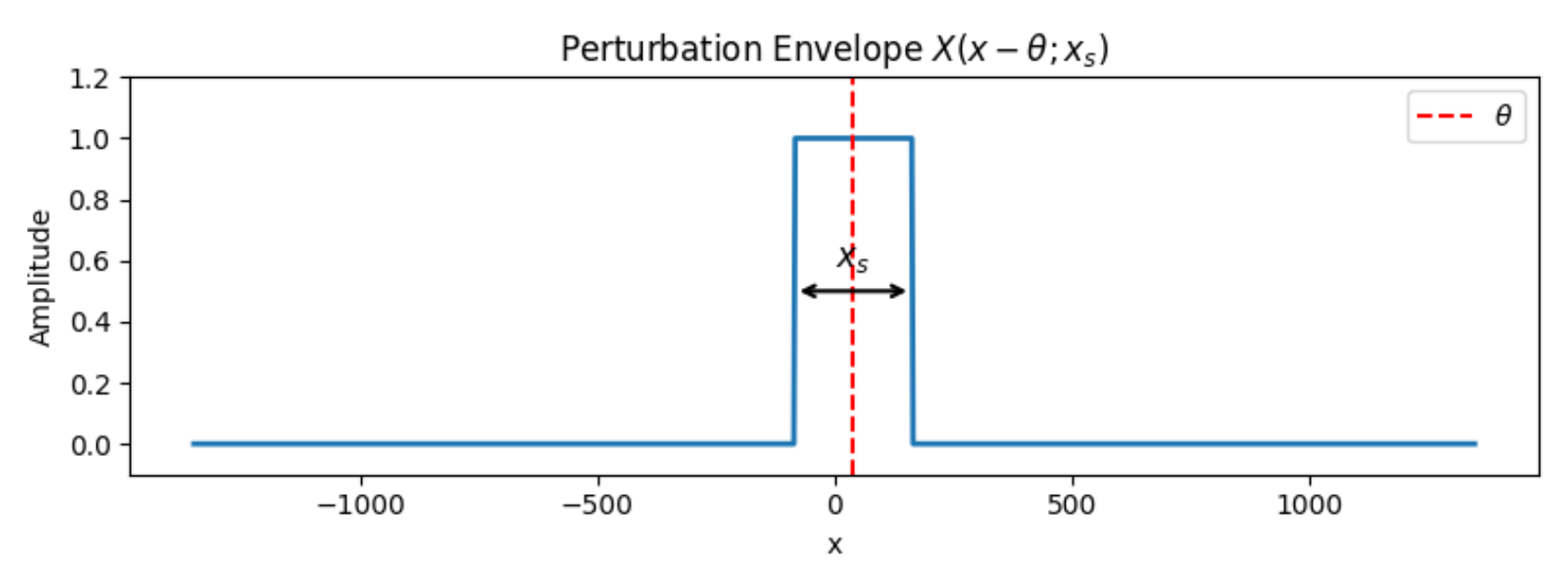


Quenched wave



Quenching threshold

- Smallest perturbation amplitude U_q^* that stops a wave
- Applied with pulse width x_s and at location θ



Computational Challenges

- Each quenching attempt needs a full PDE simulation
- Simulating as a large number of embarrassingly parallel jobs has drawbacks:
  - if number of cores $\not\approx$ number of jobs, must batch jobs
  - one job failure may result in loss of batch
  - batches subject to straggler effect

Computational Challenges

- Each quenching attempt needs a full PDE simulation
- Simulating as a large number of embarrassingly parallel jobs has drawbacks:
  - if number of cores $\not\approx$ number of jobs, must batch jobs
  - one job failure may result in loss of batch
  - batches subject to straggler effect

We need faster, smarter strategies

FitzHugh–Nagumo Model

- Described by a system of reaction-diffusion PDEs:

$$\frac{\partial u}{\partial t} = D\nabla^2 u + u(u-\beta)(1-u) - w,$$
$$\frac{\partial w}{\partial t} = \gamma(\alpha u - w),$$

  - D is the diffusion coefficient
  - $u(x,t)$ is the excitation variable (fast)
  - $w(x,t)$ is the recovery variable (slow)
  - γ timescale separation
  - α, β are model parameters

Experimental Setup

- Domain : $x \in [0, L]$, where $L = 2700$, and $t \in [0, L/2\hat{c}]$ where $\hat{c}$ is the speed of the stable traveling pulse
- Initial conditions: A traveling excitation wave perturbed by an external stimulus given by

$$u(x, 0) = u_s(x) + U_q X(x - \theta; x_s),$$

  - $u_s(x)$ is a stable traveling wave solution
  - U_q is the perturbation amplitude
  - $X(x - \theta; x_s)$ is the perturbation envelope (localized pulse)

- Periodic boundary conditions

Experimental Setup Cont.

- $\gamma \in \{0.001, 0.01, 0.02, 0.025\}$
- $\theta \in [-40, 40]$ controls the center location of the perturbation
- $x_s \in [0, 700]$ controls the width (extent) of the perturbation
- We sample:

  θ every 5 units: $-40, -35, ..., 0, ..., 35, 40$

  x_s logarithmically between 0 and 700 (91 values)
- Total combinations: $4 \times 26 \times 91 = \mathbf{6}\ |\mathbf{188}$ jobs

How We Detect Quenching

- Define $\psi(T)$: deviation from rest

$$\psi(T) = \int_0^L |u(x,T) - \bar{u}|\, dx$$

- $\bar{u}$ is the value of u at rest

What Is the Unstable Wave?

- It's a special traveling wave that sits exactly at the boundary between:
  - A wave that recovers and keeps moving
  - A wave that collapses and returns to rest
- It's not stable: any tiny deviation makes it either grow or decay

Root-Finding for Threshold

- Solve:

$$F(U_q) = \psi(T; U_q, x_s, \theta) - \psi_{\text{unstable}} = 0$$

- ψ_{unstable} is the diagnostic value of the quenching metric $\psi(T)$ when the solution approaches the unstable traveling wave.
- Use Brent method to find U_q^*:
- Repeat for each (γ, x_s, θ) pair

Baseline Algorithm (Naïve)

For each parameter pair (γ, x_s, θ)**:**

1. Set an initial bracketing interval: $U_{q^{\min}}$, $U_{q^{\max}}$
2. Run a simulation with amplitude U_q
3. Check if wave quenched
4. Use root finder to refine U_q
5. Repeat until $|F(U_q)| < \varepsilon$

Actor Model

```cpp
#include "caf/all.hpp"


caf::behavior goodbye(caf::event_based_actor* self) {
  return {
    [=](const std::string& name) {
       self->println("Goodbye, " + name + "!");
      self->quit();
    }
  };
}

struct hello_state {
  int greet_count = 0;
};

caf::behavior hello(caf::stateful_actor<hello_state>* self) {
  return {
    [=](const std::string& name) {
      ++self->state().greet_count;
      self->println("Hello, " + name + "! Count: " + std::to_string(self->state().greet_count));
      self->become(goodbye(self));
    }
  };
}

void caf_main(caf::actor_system& system) {
    caf::scoped_actor self{system};
    auto hello_actor = system.spawn(hello);
    caf::anon_mail("World").send(hello_actor);
    caf::anon_mail("Alice").send(hello_actor);

    auto goodbye_actor = system.spawn(goodbye);
    caf::anon_mail("Bob").send(goodbye_actor);
}

CAF_MAIN()
```

→ Private state

→ Become

→ Create

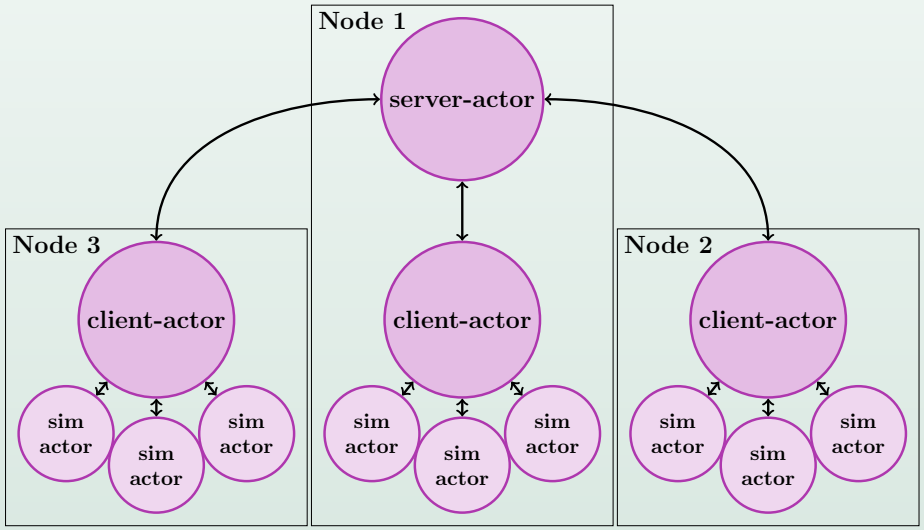
→ Communication through message

Output

```
Goodbye, Bob!
Hello, World! Count: 1
Goodbye, Alice!
```

Quenching-Actors

- **Server**: Generates jobs, manages progress
- **Client**: Spawns actors on remote nodes
- **Simulation**: Runs simulation, sends result



Smarter Bracketing Strategies

- Goal: Accelerate convergence to the quenching threshold U_q^*
- Use previously computed values to set tighter bracketing intervals:

  Same (γ, θ), nearby x_s: scale known U_q^* based on x_s
  Same (γ, x_s), smaller $|\theta|$: use U_q^* from more central positions
  Same (θ, x_s), larger γ: use U_q^* from faster waves

- Helps reduce failed root searches and shortens simulation time

Improved Algorithm (Actor-Based)

For each job (γ, θ, x_s)**:**

1. **[Server]** At first initializes job list and send jobs to client
   If there are some information in the cache, update the job list based on:

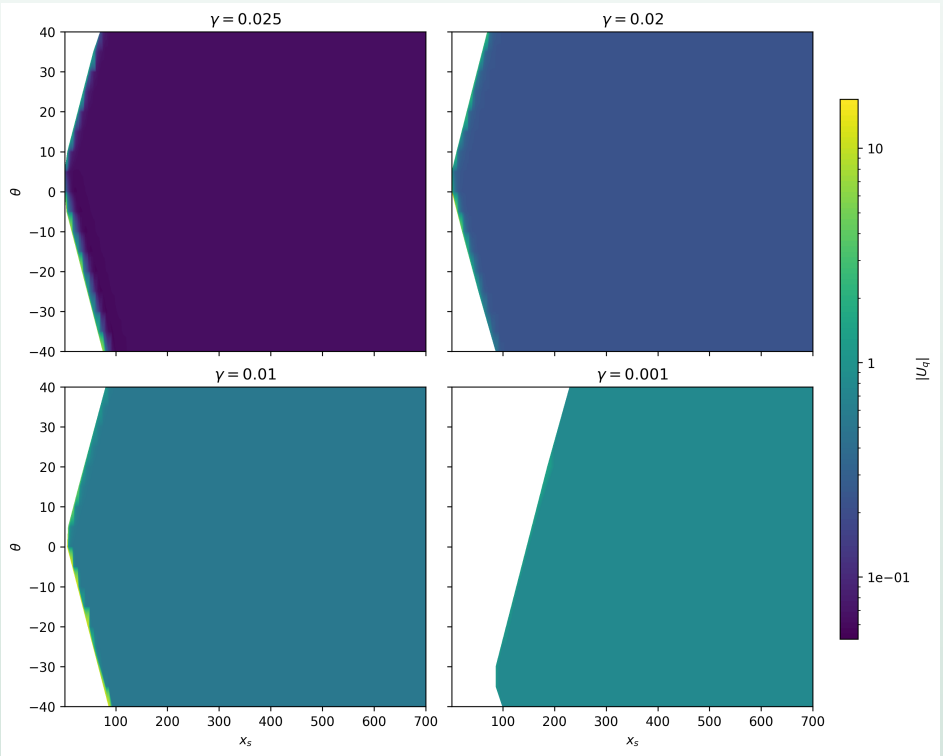
   Nearby x_s with same (γ, θ)
   Same (γ, x_s), smaller $|\theta|$
   Same (θ, x_s), larger γ

2. **[Client]** spawns actors on remote nodes
3. **[Client]** sends job to available worker
4. **[Worker]** receives a job
5. Set bracketing interval $[U_{q^{\min}}, U_{q^{\max}}]$ based on one
6. Run the solver with and send back the results
7. Save result in cache (`Uq_cache`(γ, θ))

Additional Details

- Space: 6th-order centered finite difference
- Time: BDF (`CVODE_BDF`) + GMRES (Sundials)
- The `Plato` cluster
- 5 nodes used were dual-socket Intel Xeon Gold 6226R @ 2.90GHz (Cascade Lake) processors (we used 30 cores on each node) connected to shared file system with EDR Infiniband
- More details: `https://wiki.usask.ca/display/ARC/Plato+HPC+Cluster`

Performance



- 6 188 simulations in sequence would take more than **100 days**
- Quenching-Actors without updating the brackets took **22:13:50** with 150 cores
- The CPU usage was **95.11%**

Performance Cont.

- Quenching-Actors with updating the brackets took **21:50:14** with 150 cores!
- It did not accelerate the simulation time significantly!
- Why?

Performance Cont.

- Quenching-Actors with updating the brackets took **21:50:14** with 150 cores!
- It did not accelerate the simulation time significantly!
- Why?
- The number of jobs that failed to find a quenching threshold was **3 358**!

Solution: Smolyak Sampling Instead of Exhaustive Sweeps

- **Problem:** Exhaustive sampling over (x_s, θ) is expensive, especially in regions where quenching is unlikely.
- **Observation:** Jobs that without quenching are both the slowest and the least informative.
- **Idea:** Use **Smolyak sampling** to adaptively select points that provide maximum information while reducing redundancy.

Smolyak Sampling: A Smarter Strategy (but not today)

- Smolyak sampling offers:
  - Sparse, adaptive exploration of parameter space
  - Focus near the quenching boundary
  - Fewer wasted simulations
- Do I have results yet? No!
- Will I have them next week? Probably!

Stay tuned for the sequel!

Conclusions

- The actor-based system enabled efficient parallel execution of wave quenching simulations
- Dynamic management of actors allowed for adaptive resource allocation and reduction in the burden on the user to fine-tune the simulation parameters