From 0 to 100K in 10 years:
nurturing open-source community

Bob Carpenter

Center for Computational Mathematics

Flatiron Institute

October June 2022

https://mc-stan.org

Taking uncertainty seriously

- **Uncertainty** permeates science and decision making:
  - **sampling** uncertainty
    - data is finite
  - **measurement** uncertainty
    - measurements are noisy, biased, and incomplete
  - **modeling** uncertainty
    - our models are imperfect reflections of reality

- The alternative to **good statistics** is not no statistics, but **bad statistics**. (Bill James)

Probability & Statistics

- Probability theory uses math to **quantify uncertainty**.
- **Bayesian statistics** applies probability theory to
  - data analysis
  - inference (estimation and prediction)
  - model evaluation and comparison
  - decision theory (given preferences)
- The **computational bottlenecks** for Bayes are
  - model expression
  - posterior inference

Bayesian probability is epistemic

Every event is in itself certain, not probable; if we knew all, we should either know positively that it will happen, or positively that it will not. But its **probability** to us means the **degree of expectation** of its occurrence, which we are **warranted in entertaining** by our present **evidence**.

– **John Stuart Mill**. 1882. *A System of Logic: Ratiocinative and Inductive*. Eighth edition. III:18.

Bayesian modeling

- Define a **generative model** for data y and parameters θ as a probability density $p(y \mid \theta)$
  - e.g., compose a forward scientific model and a measurement model
- Define our present evidence as a **prior** density $p(\theta)$, e.g.,
  - realistic ranges of cancer prevalence,
  - possible masses of exoplanets,
  - livable metabolic rates for humans,
  - concentration of particulates in breathable air,
  - realistic major league batting averages,
  - and so on

Bayesian inference

- Observe actual **data** y
- Calculate expectations over the **posterior**

$$p(\theta \mid y) \propto p(y \mid \theta) \cdot p(\theta)$$

  - parameter estimation $\widehat{\theta} = \mathbb{E}[\theta \mid y]$
  - forecasting events $\Pr[E \mid y] = \mathbb{E}[\mathrm{I}(\theta \in E) \mid y]$
  - predictions for new data: $p(\tilde{y} \mid y) = \mathbb{E}[p(\tilde{y} \mid \theta) \mid y]$

- Estimate with **Markov chain Monte Carlo (MCMC)** (or approximations)

What is Stan?

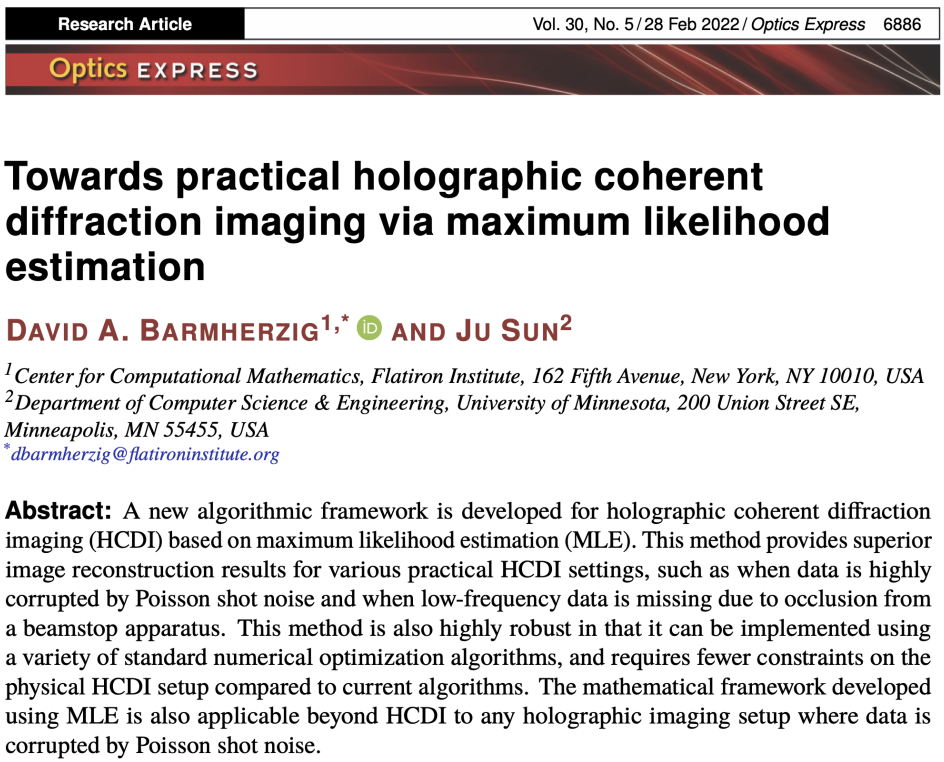
- a domain-specific **probabilistic programming language**
- Stan **program** defines a **differentiable** probability model
  - declares data and (constrained) parameter variables
  - defines log posterior (or penalized likelihood)
  - defines predictive quantities
- Stan **inference** fits model & makes predictions
  - MCMC for full Bayesian inference
  - variational and Laplace for approximate Bayes
- Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Marcus Brubaker, Jiqiang Guo, Peter Li, Riddell, A. (2017). Stan: A probabilistic programming language. *J. Stat. Soft.* 76(1).

e.g., Logistic Regression

```
data {
  int<lower=1> K;
  int<lower=0> N;
  matrix[N,K] x;
  int<lower=0,upper=1> y[N];
}
parameters {
  vector[K] beta;
}
model {
  beta ~ cauchy(0, 2.5);           // prior
  y ~ bernoulli_logit(x * beta);   // likelihood
}
```

Holographic coherent diffraction imaging

- Brian Ward (CCM) and I reproduced David Barmherzig's paper to illustrate Stan's complex number and FFT capability
- GitHub: **bob-carpenter/CDI**



Research Article Vol. 30, No. 5 / 28 Feb 2022 / Optics Express 6886

Optics EXPRESS

Towards practical holographic coherent diffraction imaging via maximum likelihood estimation

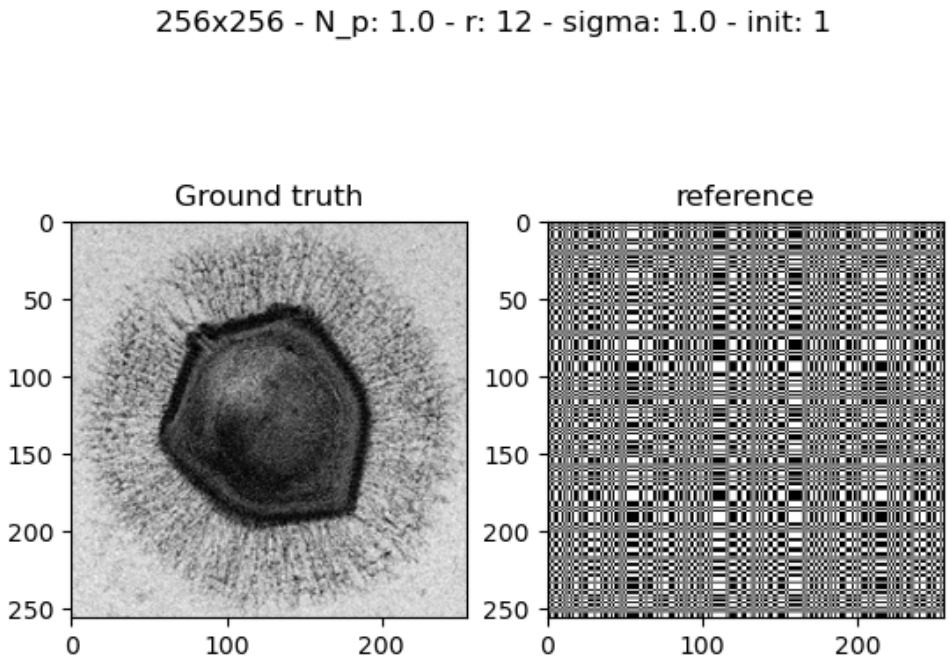
DAVID A. BARMHERZIG[1,*] AND JU SUN[2]

[1]Center for Computational Mathematics, Flatiron Institute, 162 Fifth Avenue, New York, NY 10010, USA
[2]Department of Computer Science & Engineering, University of Minnesota, 200 Union Street SE, Minneapolis, MN 55455, USA
[*]dbarmherzig@flatironinstitute.org

Abstract: A new algorithmic framework is developed for holographic coherent diffraction imaging (HCDI) based on maximum likelihood estimation (MLE). This method provides superior image reconstruction results for various practical HCDI settings, such as when data is highly corrupted by Poisson shot noise and when low-frequency data is missing due to occlusion from a beamstop apparatus. This method is also highly robust in that it can be implemented using a variety of standard numerical optimization algorithms, and requires fewer constraints on the physical HCDI setup compared to current algorithms. The mathematical framework developed using MLE is also applicable beyond HCDI to any holographic imaging setup where data is corrupted by Poisson shot noise.

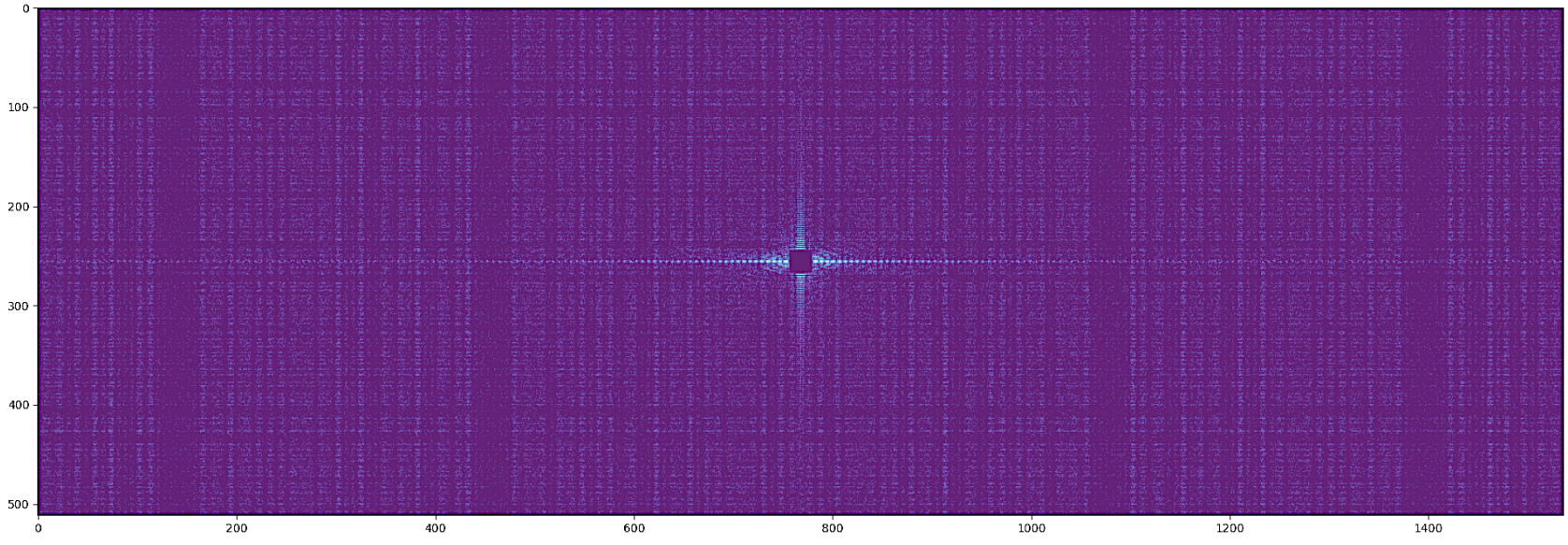
Holo CDI (1)

- Step 1: direct **coherent radiation** source (e.g., X-ray) at biomolecule and target & uniformly redundant array (URA) **reference**, with **beam stop**; (data **simulated**, but works with real)



Holo CDI (2)

- Step 2: observe diffracted **photon counts** $\tilde{Y}_{i,j}$
  - beamstop visible in middle (low frequencies)



Holo CDI (3.a)

- Step 3.a: code **data format** in Stan (simplified)

```
data {
  int<lower=0> N;                          // image dim
  matrix<lower=0, upper=1>[N, N] R;        // registration img
  int<lower=N> M1;                         // padded rows
  int<lower=3 * N> M2;                     // padded cols
  int<lower=1, upper=M1> r;                // beam stop size
  real<lower=0> N_p;                       // avg photons/pixel
  array[M1, M2] int<lower=0> Y_tilde;      // observed photons
}
```

Holo CDI (3.b)

- Step 3.b: code **transformed data, parameters, and likelihood**

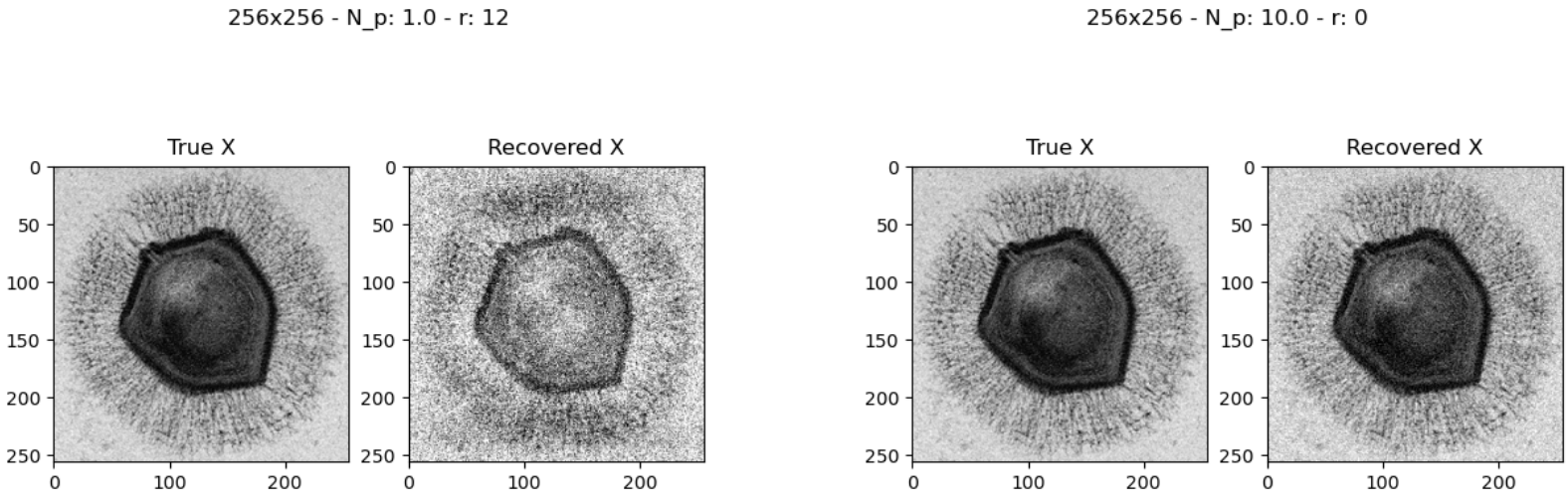
```
transformed data {
  matrix[M1, M2] B_stop = pad(M1, M2, r);  // beam stop
}
parameters {
  matrix<lower=0, upper=1>[N, N] X;        // image pixels
}
model {
  matrix[M1, M2] XOR_pad = pad(X, R, M1, M2);
  matrix[M1, M2] E_Y = B_stop .* abs(fft2(XOR_pad)) .^ 2;
  matrix[M1, M2] lambda = N_P / mean(Y) * E_Y;

  Y_tilde ~ poisson(lambda);  // likelihood
  X ~ uniform(0, 1);          // prior
}
```

Holo CDI (4)

- Turn Stan's crank to solve the **inverse problem**
- using state-of-the-art **gradient-based inference**:
  - maximum likelihood via quasi-Newton optimization (L-BFGS)
  - full Bayes via Markov chain Monte Carlo sampling (NUTS)
  - approximate Bayes via black-box variational inference (ADVI)
- For the 256×256 pixel reconstructions in David's paper,
  - optimization solves inverse problem in **2 minutes**
  - sampling solves inverse problem in **2 hours**
- Full study considers L1 & L2 regularizing **spatial priors** (penalize diffs in adjacent pixels)

Holo CDI (5.a)

- Recovered images X^* at 256×256 with
  1 photon/pixel (left) vs. 10 photons/pixel (right)



- MLE: $X^* = \arg\max_X \, p(X|\tilde{Y}, N, R, r)$
- Bayesian estimate: $\hat{X} = \mathbb{E}[X \mid \tilde{Y}, N, R, r]$

Availability & Usage

- *Platforms:* Linux, Mac OS X, Windows
- *Interfaces:* R, Python, Julia, MATLAB, Shell, C++
- *Developers (academia & industry):* 40+ (10+ FTEs)
- *Users:* hundreds of thousands
- *Companies using:* hundreds or thousands
- *Downloads:* millions
- *User's Group:* 5000+ registered; 30 posts & 20K views/day
- *Books using:* 10+
- *Courses using:* 100+ (5+ on YouTube)
- *Case studies about:* 100+
- *Articles using:* 20,000+ (500+ just on Covid!)
- *Conferences:* 4 (800+ attendance)

Some published applications

- **Physical sciences**: astrophysics, statistical mechanics, particle physics, (organic) chemistry, geology, oceanography, climatology, biogeochemistry, materials science, ...
- **Biological sciences**: molecular biology, clinical drug trials, entomology, pharmacology, toxicology, opthalmology, neurology, genomics, agriculture, botany, fisheries, epidemiology, population ecology, neurology, psychiatry, ...
- **Social sciences**: econometrics (macro and micro), population dynamics, cognitive science, psycholinguistics, social networks, political science, survey sampling, anthropology, sociology, social work, ...
- **Other**: education, public health, A/B testing, government, finance, machine learning, logistics, electrical engineering, transportation, actuarial science, sports, advertising, marketing, ...

Industries using Stan

- **marketing attribution**: Google, Domino's Pizza, Legendary Ent.
- **demand forecasting**: Facebook, Salesforce
- **financial modeling**: Two Sigma, Point72
- **pharmacology & CTs**: Novartis, Pfizer, Astra Zeneca
- **(e-)sports analytics**: Tampa Bay Rays, NBA, Sony Playstation
- **survey sampling**: YouGov, Catalist
- **agronomy**: Climate Corp., CiBO Analytics
- **real estate pricing models**: Reaktor
- **industrial process control**: Fero Labs

Why is Stan so Popular?

- **Community**: large, friendly, helpful, and sharing
- **Documentation**: novice to expert; breadth of fields
- **Robustness**: industrial-strength code; user diagnostics
- **Flexibility**: highly expressive language; large math lib
- **Portability**: popular OS, language, and cloud support
- **Extensibility**: developer friendly; derived packages
- **Speed**: $2 - \infty$ orders of magnitude faster
- **Scalability**: orders of magnitude more scalable than previous
- **Openness**: permissive code and doc licensing

Start with a Real Motivation

- a reason for people to use your project (speed, scalability, ease of use, portability, robustness, functionality, etc.)
- if you alleviate a pain point, your tool will be used
- . . . as long as it doesn't cause more collateral pain
- January 2010: Andrew Gelman can't fit the models in his and Jennifer Hill's hierarchical regression book in BUGS or express them in lme4
- so he hires two computer scientists with industrial coding experience (me & Matt Hoffman) to try to figure it out
- we failed to do that for several reasons . . .

Faster Horses

- There's an apocryphal story that Henry Ford once said that if he'd asked his customers what they wanted, they'd have said **faster horses**.
- the point is **not to ignore your customers**
- it's to **figure out what they really want** (faster, easier travel)
- After souping up JAGS (C++ version of BUGS), Matt realized better engineering on JAGS was like breeding faster horses
- After studying lme4, I realized we needed a more expressive language than a generalized lme4

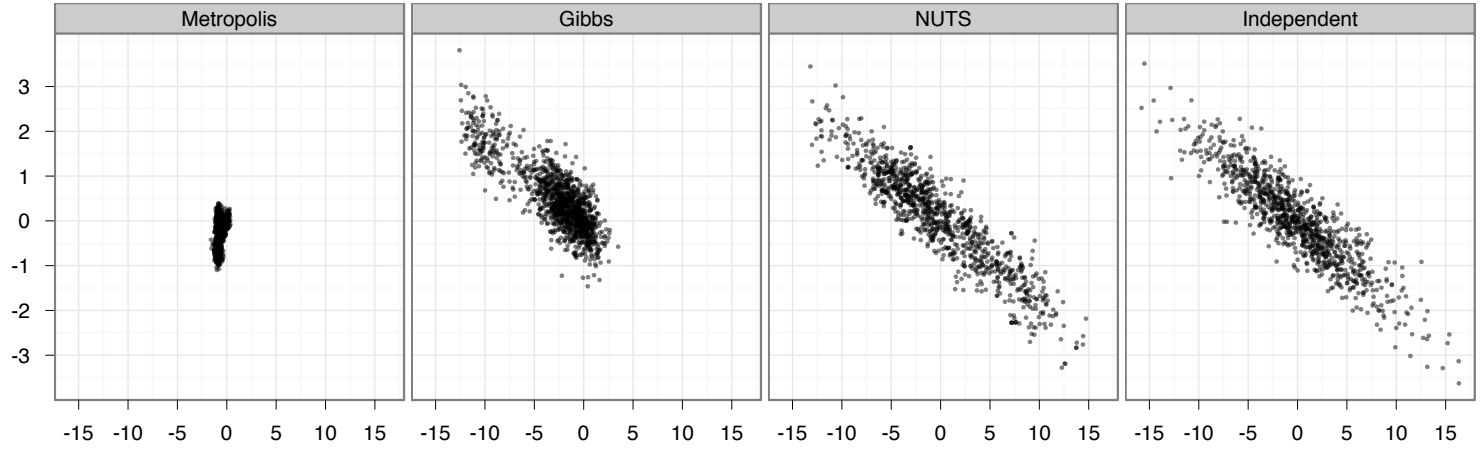
Step 1: Ask the Community

- Andrew has a ridiculously popular blog, *Statistical Statistical Modeling, Causal Inference, and Social Science*
- So I asked for help in terms of where to look for more scalable samplers
- Word on the street was Hamiltonian Monte Carlo
- but it's almost impossible to tune
- and it requires gradients

Pain Point 2: Tuning HMC

- HMC is really really hard to tune
- this time, nobody had any good ideas
- Andrew formulated the goal as maximizing expected square jump distance and tuning acceptance
- Matt started thinking about the U-turn idea to avoid waste
- then he worked out how to maintain detailed balance
- and add bias to get draws close to half an orbit
- this was all before we had any users

NUTS vs. Gibbs and Metropolis samplers



- Two dimensions of highly correlated 250-dim normal
- **1,000,000 draws** from Metropolis and Gibbs (thin to 1000)
- **1000 draws** from NUTS; 1000 independent draws
- HMC is $\mathcal{O}(N^{5/4})$ in dimension vs. Gibbs/Metropolis $\mathcal{O}(N^2)$

Pain Point 3: Derivatives

- I blegged again for help in computing gradients
- Word on the street was automatic differentiation
  - we thought they meant finite differences
  - but did a Wikipedia search and learned it was generalized backpropagation

Autodiff engineering

- this was before TensorFlow, PyTorch, or JAX
- existing tools in 2011 were not well engineered
- for either dynamic gradient performance or extension
- Matt and I put our industry background to use and engineered a better C++ version
- using a modern lazy adjoint-Jacobian design
- I named the first repo agrad

Horses for courses

- after TensorFlow and PyTorch hit the scene, we started planning for being overtaken
- . . . but it still hasn't happened
- big neural networks can exploit SIMD instructions, typical statistical models cannot
- that gives us very different optimization goals
- they're much faster on GPU and we're much faster on CPU (as reported by Google's TensorFlow team)
- still improving (most recently, matrix memory locality)

Identify a point of entry

- You need to solve at least one problem well
- For us, it was hierarchical generalized linear models.
- Just better solvers for those would be a big win
- Work very hard and focus on an early win
  - for Stan, it was a PK/PD non-linear mixed effects model for a Phase I clinical trial with Novartis
- Daniel and I spent 6 months optimizing C++ (template traits, vectorizing, expression templates, checkpointing, analytic gradient unfolding) until Stan was faster than JAGS on its own example models (all small scale)

It can be incremental

- Most projects are not very innovative
- Our autodiff is a better engineered version of Trilinos Sacado
- Our language is a generalized, typed version of BUGS
- Our sampler is an adaptive form of HMC (OK, that's innovative)

Your grain of salt

- We did a lot of this consciously, but this report is pure hindsight
- If you haven't taken your grain of salt yet, please do
- It's hard to admit, but ventures like this require **a lot of luck**, especially **good timing**

Marketing and sales

- Your project won't sell itself any more than your paper will
- We have Andrew Gelman's bully pulpit—the blog
  - daily blog posts get 10K+ views including top statisticians
  - Andrew also wrote the standard textbook and updated it to Stan
- When Gelman puts you in his textbooks and blogs about you non-stop, people in the Bayes comp stats world listen
- We also did lots of meetups, lectures, hackathons, etc. to get the word out

Why hadn't it been done before?

- HMC is hard to tune
- Hand-derived gradients are too big an ask
- Autodiff at the time was primitive
- Language and compiler design is hard
- Constrained parameters are painful for HMC
  - Stan automates changes of variables
  - gradients of log absoute Jacobian determinants
  - working with Meenal Jhajaria (CCM intern) this summer comparing transform statistical & computational efficiency

Assemble the right team

- Previous efforts used statisticians exploring limited applications and with no CS experience
  - epidemiologists built BUGS and JAGS
  - the BUGS devs said on record that if they could code C++, they'd have gone into finance
  - R was a terrible role model as language & community
- Our project started day one with two **crack statisticians** (Gelman and Ben Goodrich),
- plus three **industrially trained computer scientists** (me and Matt plus Daniel Lee)
- and two years of financial runway in the form of grants

Reassemble the team

- Grad students and postdocs come and go
- So do software engineers once trained (industry pays better)
- Need to keep recruiting devs to ensure project continuity
- Pair programming is great for bringing up to speed
- Tutorial onboarding docs are also critical
- As is being welcoming and prioritizing new contributions

Support individual users

- **Documentation** is necessary, but **not sufficient**
- Need to **help individuals** on message boards, at meetups, ...
  - make sure user queries messages get answered in a timely fashion
  - if they can't install your software or figure out how to use it, they will go away
- Helping on message boards is rewarding (also learn a lot)
- But it's very draining until it **bootstraps to self-sustaining** (thousands of visitors)

Go with an open license

- This used to be more contentious
- MIT- and BSD-like licenses seem to have won over copyleft licenses like GPL
- Everyone asks why we give it away free to industry
  - they give back in cash and in kind
  - and your devs may wind up there soon and still want to use the system they built
  - cool open source spinoffs like Prophet

Code Review

- A tech manager friend of mine asked how we did code review.
- I admitted we didn't.
- Adding that was the single biggest quality boost we made
- Two eyes are way better than one at spotting flaws
- With review, at least two people understand each piece of code
- It also keeps you honest knowing there will be a review

Earn trust through openness

- Transparency and open discussion leads to trust (GitHub, forums, blog posts, etc.)
- Find a way to be responsive
- Underpromise and overdeliver
- Admit errors, advertise and take responsibility (and patch!) bugs ASAP
- Similar projects are your allies, not your enemies
- If people understand your problems, they'll cut you slack
  - but things can be hard to explain to non-specialists (stats or CS)

Be very careful with comparisons

- If your project has advantages, you won't need detailed comparisons
- Comparing adaptive stochastic systems is a **Very Hard Thing**
- Be gracious to your competitors and they'll be gracious to you

Your project depends on evangelists

- You need to help the early adopters and they'll help others
- Early adopters often become evangelists or devs
- Don't be surprised when the evangelists oversell your project
- User tutorials at conferences are good venues
- Longer hands-on tutorials and multi-session classes are even better

Grow your process with your project

- Move from Daniel and me back to back in an office
- to dozens of devs communicating through forum posts, Slack, GitHub and small Zoom or in-person meetings
- Need to keep continuous integration testing up to date
- We grew into all this organically as the project grew.
- We now have fancy things like syntax highlighting in mark-down and Git and are even a language reported by GitHub

Support others building on your tooling

- Hundreds of other packages built on Stan in R and Python
- Derived projects are used more than Stan itself
- It'll make your software better and expand your user base
- Support other groups trying to build a better mousetrap
  - learn from them, or plan your project's retirement
  - either way, it's good

High-level interfaces

- To my surprise, some people find Stan challenging
- For them, we have high-level interfaces
- Most interfaces encapsulate whole models and data ingestion formats
  - Facebook built Prophet, a demand forecasting system
  - Metrum built Torsten, a pharmacology modeling system
  - others have distributed models (like Google's ad attribution)
- Others encapsulate whole classes of models (rstanarm) or introduce sublanguages (brms)

Governance

- Politics (group decision making) is hard
- Want zone before techie zeal and academic reticence
- I resisted being made benevolent dictator for life
- But wrongly instituted module decision makers and a tech director
- No siree, Bob. You really want to follow Apache and go with voting
- which pushes politics to who votes on what

Scaling Out

- Scaling up (at one institution) only goes so far
- Embrace scaling out
- We have “centers” in Helsinki, Ljubljana, and now two in NYC with Flatiron and Columbia
- Support fully remote workers (we got used to online meets before Covid)

Measuring community is hard

- Too many distribution channels
- Too many bots and multiple downloads
- What's a user?
  - current or one-time use? classroom use?
  - do you have to write the code or just run it?
  - what about embedded uses?
- Proxies include citations, books, classes, videos, stars on GitHub, contributing devs, user group clicks, etc.
- or mentions in the mainstream media

Naming

- I named the initial repo agrad
- Andrew really settled on Stan (for Stanislaw Ulam [and the Eminem song])
- Hadley Wickham and I tried to convince him we wanted
  - zero hits on Google
  - not spell corrected
- The name “Stan” is unsearchable
- but it’s growing on me

Questions?

- To learn more, see: `https://mc-stan.org`

Music and Cocktails?

- I hear Alex has assembled a band
- and Jonathan is making Stanleys on the roof