Variational inference for stochastic differential equations

Dennis Prangle

Newcastle University, UK

October 2018

Acknowledgements and reference

- Joint work with Tom Ryder, Steve McGough, Andy Golightly



- Supported by EPSRC cloud computing for big data CDT
- and NVIDIA academic GPU grant
- Published in ICML 2018
- http://proceedings.mlr.press/v80/ryder18a.html
- https://github.com/Tom-Ryder/VIforSDEs

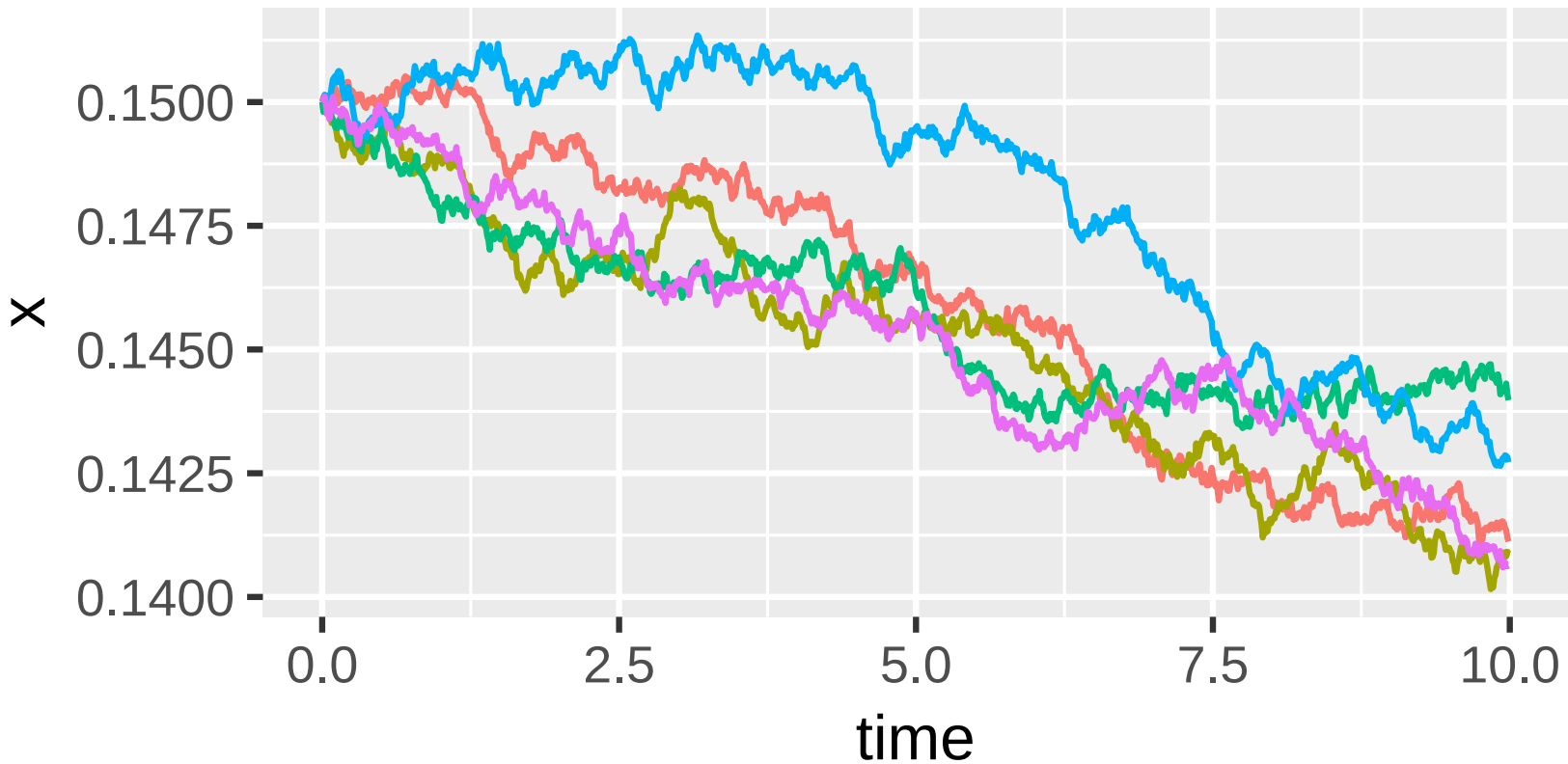
Overview

- Background
  - Stochastic differential equations
  - Variational inference
- Variational inference for SDEs
- Example
- Conclusion

Stochastic differential equations (SDEs)

SDEs

- SDE perturbs a differential equation with random noise
- Defines a **diffusion process**
- Function(s) which evolves randomly over time
- SDE describes its **instantaneous** behaviour



SDE applications

- Finance/econometrics (Black & Scholes, 1973; Eraker, 2001)
- Biology/ecology (Gillespie, 2000; Golightly & Wilkinson, 2011)
- Physics (van Kampen, 2007)
- Epidemiology (Fuchs, 2013)

Univariate SDE definition

$$dX_t = \alpha(X_t, \theta)dt + \sqrt{\beta(X_t, \theta)}dW_t, \quad X_0 = x_0.$$

- X_t is random variable at time t
- α is **drift** term
- β is **diffusion** term
- W_t is Brownian motion process
- θ is unknown parameters
- x_0 is initial conditions (can depend on θ)
- Formalisation requires **stochastic calculus** (e.g. Itô)

Univariate SDE definition

$$dX_t = \alpha(X_t, \theta)dt + \sqrt{\beta(X_t, \theta)}dW_t, \quad X_0 = x_0.$$

- X_t is random variable at time t
- α is **drift** term
- β is **diffusion** term
- W_t is Brownian motion process
- θ is unknown parameters
- x_0 is initial conditions (can depend on θ)
- Formalisation requires **stochastic calculus** (e.g. Itô)

Univariate SDE definition

$$dX_t = \alpha(X_t, \theta)dt + \sqrt{\beta(X_t, \theta)}dW_t, \quad X_0 = x_0.$$

- X_t is random variable at time t
- α is **drift** term
- β is **diffusion** term
- W_t is Brownian motion process
- θ is unknown parameters
- x_0 is initial conditions (can depend on θ)
- Formalisation requires **stochastic calculus** (e.g. Itô)

Multivariate SDE definition

$$dX_t = \alpha(X_t, \theta)dt + \sqrt{\beta(X_t, \theta)}dW_t, \quad X_0 = x_0.$$

- X_t is random vector
- α is drift vector
- β is diffusion matrix
- W_t is Brownian motion process vector
- θ is unknown parameters
- x_0 is initial conditions

Problem statement

- We observe diffusion at several time points
- (Usually partial noisy observations)
- **Primary goal**: infer parameters θ
- e.g. their posterior distribution
- **Secondary goal**: also infer diffusion states x

Problem statement

- We observe diffusion at several time points
- (Usually partial noisy observations)
- **Primary goal**: infer parameters θ
- e.g. their posterior distribution
- **Secondary goal**: also infer diffusion states x

SDE discretisation

- Hard to work with exact SDEs
- Common approach is **discretisation**
- Approximate on a discrete grid of times
  e.g. evenly spaced $0, \Delta\tau, 2\Delta\tau, 3\Delta\tau \ldots$
- Simplest method is **Euler-Maruyama**

Euler-Maruyama discretisation

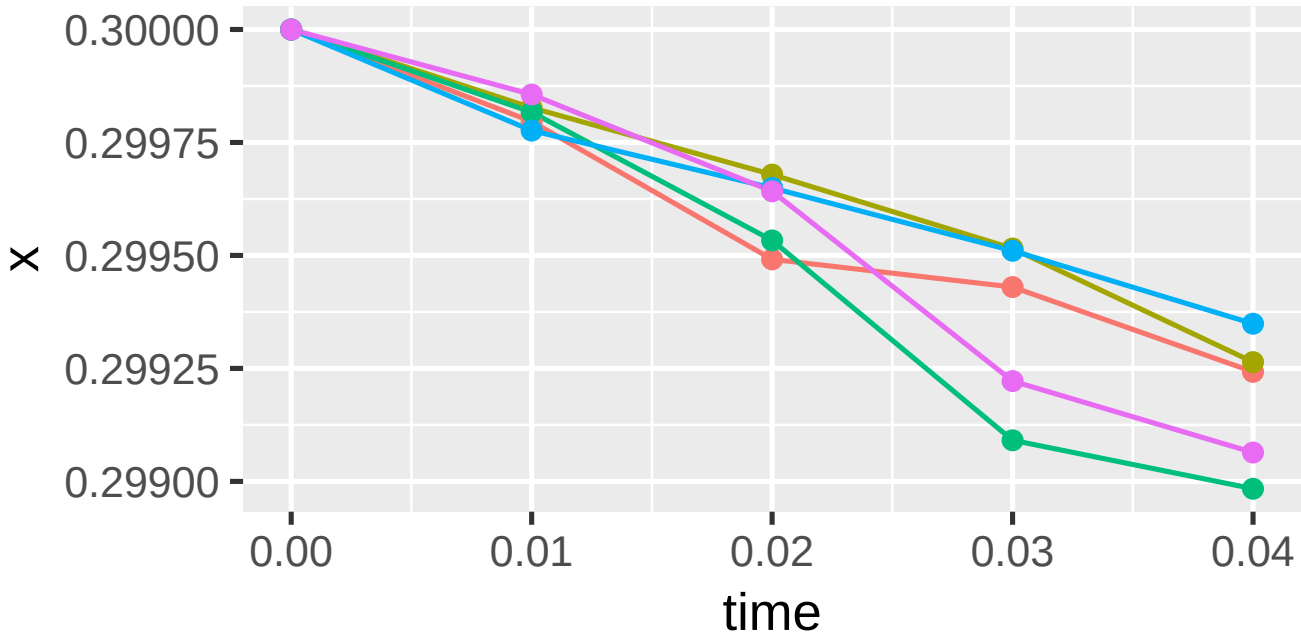
$$x_{i+1} = x_i + \alpha(x_i, \theta)\Delta\tau + \sqrt{\beta(x_i, \theta)\Delta\tau}\, z_{i+1}$$

- x_i is state at ith time in grid
- $\Delta\tau$ is grid timestep
- z_{i+1} is vector of independent $N(0, 1)$ draws

Euler-Maruyama discretisation

$$x_{i+1} = x_i + \alpha(x_i, \theta)\Delta\tau + \sqrt{\beta(x_i, \theta)\Delta\tau}\, z_{i+1}$$

- x_i is state at ith time in grid
- $\Delta\tau$ is grid timestep
- z_{i+1} is vector of independent $N(0, 1)$ draws



Posterior distribution

- Let $p(\theta)$ be prior density for parameters
- Posterior distribution is

$$p(\theta, x|y) \propto p(\theta)p(x|\theta)p(y|x,\theta)$$

(prior $\times$ SDE model $\times$ observation model)

- where
  - $p(x|\theta)$ product of normal densities for state increments (i.e. $x_{i+1} - x_i$ values)
  - and $p(y|x,\theta)$ product of normal densities at observation times
- n.b. right hand side is **unnormalised** posterior $p(\theta, x, y)$

Posterior distribution

- Let $p(\theta)$ be prior density for parameters
- Posterior distribution is

$$p(\theta, x|y) \propto p(\theta)p(x|\theta)p(y|x,\theta)$$

  (prior $\times$ SDE model $\times$ observation model)
- where
  - $p(x|\theta)$ product of normal densities for state increments (i.e. $x_{i+1} - x_i$ values)
  - and $p(y|x,\theta)$ product of normal densities at observation times
- n.b. right hand side is **unnormalised** posterior $p(\theta, x, y)$

Posterior distribution

- Let $p(\theta)$ be prior density for parameters
- Posterior distribution is

$$p(\theta, x|y) \propto p(\theta)p(x|\theta)p(y|x,\theta)$$

  (prior $\times$ SDE model $\times$ observation model)
- where
  - $p(x|\theta)$ product of normal densities for state increments (i.e. $x_{i+1} - x_i$ values)
  - and $p(y|x,\theta)$ product of normal densities at observation times
- n.b. right hand side is **unnormalised** posterior $p(\theta, x, y)$

Posterior inference

- Could use sampling methods
- e.g. MCMC (Markov chain Monte Carlo), SMC (sequential Monte Carlo)
- But posterior high dimensional and lots of dependency
- Very challenging for sampling methods
- One approach is to use bridging (next slide)

Bridge constructs

- Propose x via a **bridge construct**
- Use within Monte Carlo inference
- Bridge construct is approx to conditioned diffusion
  (usually conditioning just on next observation)
- Derived mathematically
- Various bridges used in practice
- Struggle with highly non-linear paths and large gaps between observation times
- Choosing bridges and designing new ones hard work!
- We automate this using machine learning

Bridge constructs

- Propose x via a **bridge construct**
- Use within Monte Carlo inference
- Bridge construct is approx to conditioned diffusion
  (usually conditioning just on next observation)
- Derived mathematically
- Various bridges used in practice
- Struggle with highly non-linear paths and large gaps between observation times
- Choosing bridges and designing new ones hard work!
- We automate this using machine learning

Bridge constructs

- Propose x via a **bridge construct**
- Use within Monte Carlo inference
- Bridge construct is approx to conditioned diffusion
  (usually conditioning just on next observation)
- Derived mathematically
- Various bridges used in practice
- Struggle with highly non-linear paths and large gaps between observation times
- Choosing bridges and designing new ones hard work!
- We automate this using machine learning

Variational inference

Variational inference

- Goal: inference on posterior $p(\theta|y)$
  - Given **unnormalised** version $p(\theta, y)$
- Introduce $q(\theta; \phi)$
  - Family of approximate posteriors
  - Controlled by parameters ϕ
- Idea: find ϕ giving best approximate posterior
- Converts Bayesian inference into **optimisation** problem
- n.b. outputs approximation to posterior

Variational inference

- Goal: inference on posterior $p(\theta|y)$
  - Given **unnormalised** version $p(\theta, y)$
- Introduce $q(\theta; \phi)$
  - Family of approximate posteriors
  - Controlled by parameters ϕ
- Idea: find ϕ giving best approximate posterior
- Converts Bayesian inference into **optimisation** problem
- n.b. outputs approximation to posterior

Variational inference

- Goal: inference on posterior $p(\theta|y)$
  - Given **unnormalised** version $p(\theta, y)$
- Introduce $q(\theta; \phi)$
  - Family of approximate posteriors
  - Controlled by parameters ϕ
- Idea: find ϕ giving best approximate posterior
- Converts Bayesian inference into **optimisation** problem
- n.b. outputs approximation to posterior

Variational inference

- Goal: inference on posterior $p(\theta|y)$
  - Given **unnormalised** version $p(\theta, y)$
- Introduce $q(\theta; \phi)$
  - Family of approximate posteriors
  - Controlled by parameters ϕ
- Idea: find ϕ giving best approximate posterior
- Converts Bayesian inference into **optimisation** problem
- n.b. outputs approximation to posterior

Variational inference

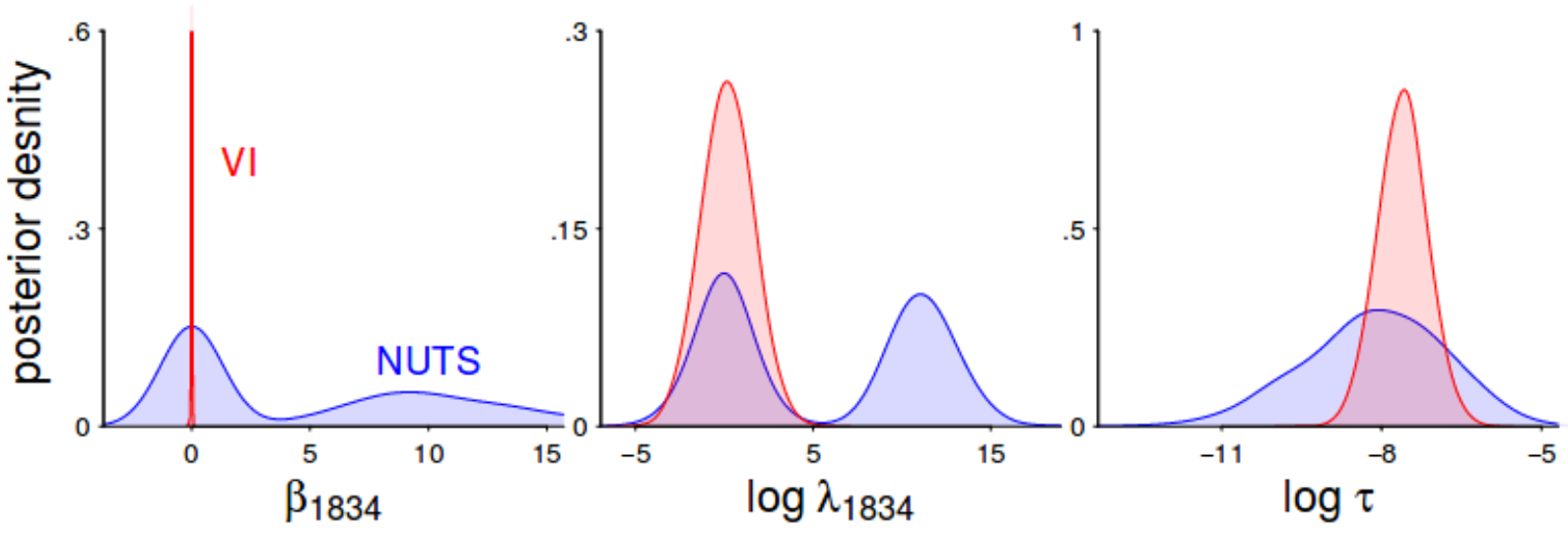
- VI finds ϕ minimising $KL(q(\theta;\phi)||p(\theta|y))$
- Equivalent to maximising **ELBO** (evidence lower bound),

$$\mathcal{L}(\phi) = \mathbb{E}_{\theta\sim q(\cdot;\phi)}\left[\log \frac{p(\theta, y)}{q(\theta;\phi)}\right]$$

(Jordan, Ghahramani, Jaakkola, Saul 1999)

Variational inference

- Optimum q often finds posterior mode well
- But usually overconcentrated!
  (unless family of qs allows very good matches)



(source: Yao, Vehtari, Simpson, Gelman 2018)

Maximising the ELBO

- Several optimisation methods:
  - Variational calculus
  - Parametric optimisation (various flavours)

Maximising the ELBO

- "Reparameterisation trick"
  (Kingma and Welling 2014; Rezende, Mohamed and Wierstra 2014; Titsias and Lázaro-Gredilla 2014)
- Write $\theta \sim q(\cdot; \phi)$ as $\theta = g(\epsilon, \phi)$ where
  - g inverible function
  - ϵ random variable of fixed distribution
- Example shortly!

Maximising the ELBO

- ELBO is

$$\mathcal{L}(\phi) = \mathbb{E}_\epsilon \left[\log \frac{p(\theta, y)}{q(\theta; \phi)} \right]$$

$$\Rightarrow \nabla_\phi \mathcal{L}(\phi) = \mathbb{E}_\epsilon \left[\nabla_\phi \log \frac{p(\theta, y)}{q(\theta; \phi)} \right]$$

- Unbiased Monte-Carlo gradient estimate

$$\nabla_\phi \hat{\mathcal{L}}(\phi) = \nabla_\phi \log \frac{p(\theta, y)}{q(\theta; \phi)}$$

- where $\theta = g(\epsilon, \phi)$ for some ϵ sample
  (can average batch of estimates to reduce variance)
- Get gradients using automatic differentiation
- Optimise $\mathcal{L}(\phi)$ by stochastic optimisation
- Easy to code in Tensorflow, PyTorch etc

Maximising the ELBO

- ELBO is

$$\mathcal{L}(\phi) = \mathbb{E}_\epsilon \left[\log \frac{p(\theta, y)}{q(\theta; \phi)} \right]$$

$$\Rightarrow \nabla_\phi \mathcal{L}(\phi) = \mathbb{E}_\epsilon \left[\nabla_\phi \log \frac{p(\theta, y)}{q(\theta; \phi)} \right]$$

- Unbiased Monte-Carlo gradient estimate

$$\nabla_\phi \hat{\mathcal{L}}(\phi) = \nabla_\phi \log \frac{p(\theta, y)}{q(\theta; \phi)}$$

- where $\theta = g(\epsilon, \phi)$ for some ϵ sample
  (can average batch of estimates to reduce variance)
- Get gradients using automatic differentiation
- Optimise $\mathcal{L}(\phi)$ by stochastic optimisation
- Easy to code in Tensorflow, PyTorch etc

Maximising the ELBO

- ELBO is

$$\mathcal{L}(\phi) = \mathbb{E}_\epsilon \left[\log \frac{p(\theta, y)}{q(\theta; \phi)} \right]$$

$$\Rightarrow \nabla_\phi \mathcal{L}(\phi) = \mathbb{E}_\epsilon \left[\nabla_\phi \log \frac{p(\theta, y)}{q(\theta; \phi)} \right]$$

- Unbiased Monte-Carlo gradient estimate

$$\nabla_\phi \hat{\mathcal{L}}(\phi) = \nabla_\phi \log \frac{p(\theta, y)}{q(\theta; \phi)}$$

- where $\theta = g(\epsilon, \phi)$ for some ϵ sample
  (can average batch of estimates to reduce variance)
- Get gradients using automatic differentiation
- Optimise $\mathcal{L}(\phi)$ by stochastic optimisation
- Easy to code in Tensorflow, PyTorch etc

Maximising the ELBO

- ELBO is

$$\mathcal{L}(\phi) = \mathbb{E}_\epsilon \left[\log \frac{p(\theta, y)}{q(\theta; \phi)} \right]$$

$$\Rightarrow \nabla_\phi \mathcal{L}(\phi) = \mathbb{E}_\epsilon \left[\nabla_\phi \log \frac{p(\theta, y)}{q(\theta; \phi)} \right]$$

- Unbiased Monte-Carlo gradient estimate

$$\nabla_\phi \hat{\mathcal{L}}(\phi) = \nabla_\phi \log \frac{p(\theta, y)}{q(\theta; \phi)}$$

- where $\theta = g(\epsilon, \phi)$ for some ϵ sample
  (can average batch of estimates to reduce variance)
- Get gradients using automatic differentiation
- Optimise $\mathcal{L}(\phi)$ by stochastic optimisation
- Easy to code in Tensorflow, PyTorch etc

Example: mean field approximation

- Simplest variational approximation
- Assumes $\theta \sim N(\mu, \Sigma)$ for Σ diagonal
- Then $\theta = g(\epsilon, \phi) = \mu + \Sigma^{1/2}\epsilon$
- where:

$$\epsilon \sim N(0, \mathrm{I})$$
$$\phi = (\mu, \Sigma)$$

- Makes strong unrealistic assumptions about posterior!

Example: mean field approximation

- Simplest variational approximation
- Assumes $\theta \sim N(\mu, \Sigma)$ for Σ diagonal
- Then $\theta = g(\epsilon, \phi) = \mu + \Sigma^{1/2}\epsilon$
- where:

  $\epsilon \sim N(0, \mathrm{I})$
  $\phi = (\mu, \Sigma)$

- Makes strong unrealistic assumptions about posterior!

Example: mean field approximation

- Simplest variational approximation
- Assumes $\theta \sim N(\mu, \Sigma)$ for Σ diagonal
- Then $\theta = g(\epsilon, \phi) = \mu + \Sigma^{1/2}\epsilon$
- where:

  $\epsilon \sim N(0, \mathrm{I})$
  $\phi = (\mu, \Sigma)$
- Makes strong unrealistic assumptions about posterior!

Variational inference: summary

- Define family of approximate posteriors $q(\theta; \phi)$
- So that $\theta = g(\epsilon, \phi)$
- Can use optimisation to minimise KL divergence
- Output is approximate posterior
- (often good point estimate but overconcentrated)

Variational inference for SDEs

Variational inference for SDEs

- We want posterior $p(\theta, x|y)$ for SDE model
- Define

$$q(\theta, x; \phi) = q(\theta; \phi_\theta) q(x|\theta; \phi_x)$$

- We use mean-field approx for $q(\theta; \phi_\theta)$
- Leaves choice of $q(x|\theta; \phi_x)$

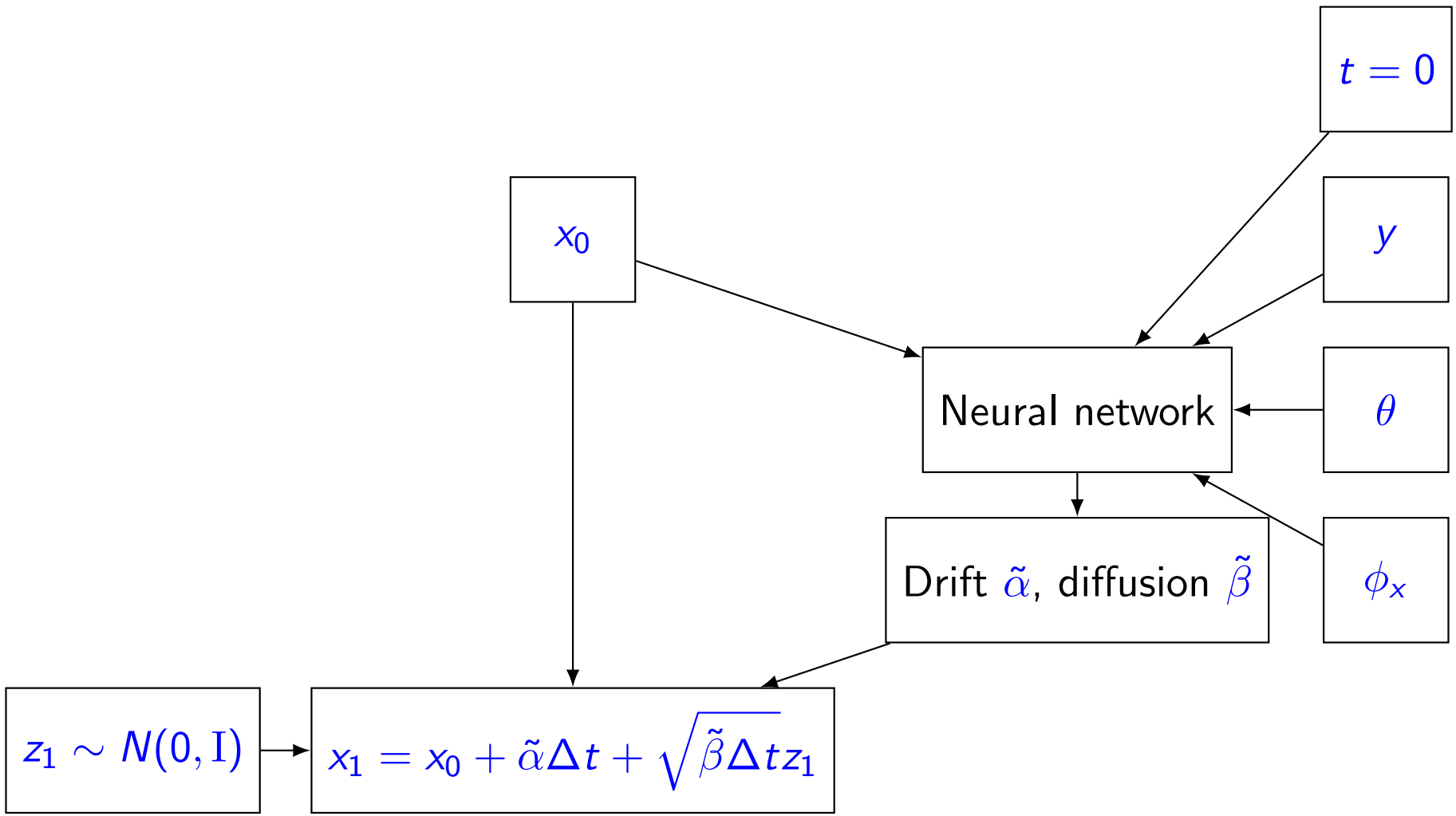
Variational inference for SDEs

- $q(x|\theta;\phi_x)$ should approximate $p(x|\theta,y)$: "conditioned diffusion"
- SDE theory suggests this is itself a diffusion (see e.g. Rogers and Williams 2013)
- But with different drift and diffusion to original SDE

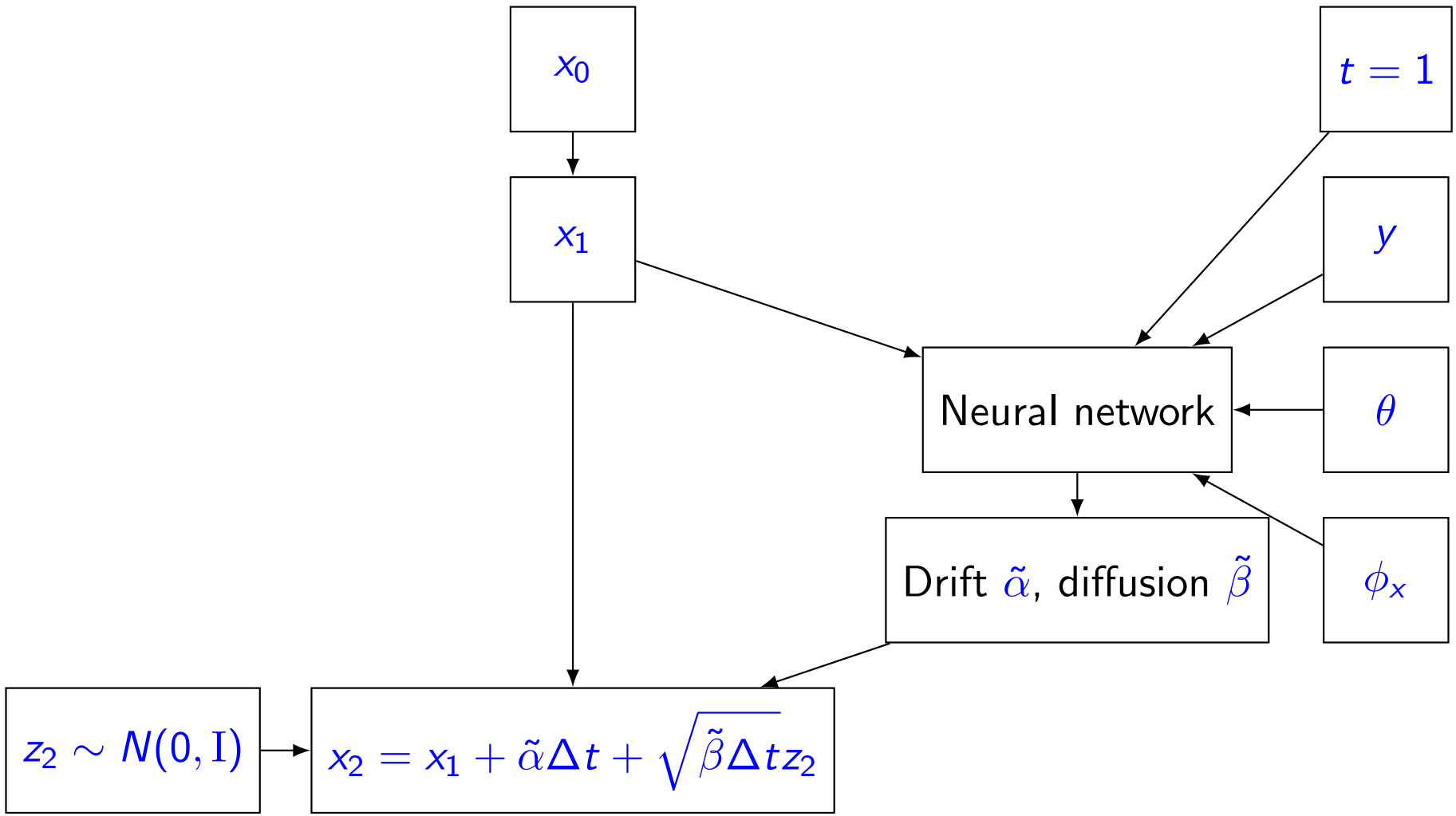
Variational approximation to diffusion

- We define $q(x|\theta;\phi_x)$ to be a diffusion
- We let drift $\tilde{\alpha}$ and diffusion $\tilde{\beta}$ depend on:
  - Parameters θ
  - Most recent x and t values
  - Details of next observations
- To get flexible parametric functions we use **neural network**
- ϕ_x is neural network parameters (weights and biases)

Variational approximation to diffusion



Variational approximation to diffusion



Variational approximation to diffusion

- Typically we take

$$x_{i+1} = x_i + \tilde{\alpha}\Delta t + \sqrt{\tilde{\beta}\Delta t} z_{i+1}$$

- Sometimes want to ensure non-negativity of xs
- So we use

$$x_{i+1} = h\left(x_i + \tilde{\alpha}\Delta t + \sqrt{\tilde{\beta}\Delta t} z_{i+1}\right)$$

- Where h outputs non-negative values e.g. softplus function

$$h(z) = \log(1 + e^z)$$

Variational approximation to diffusion

- Typically we take

$$x_{i+1} = x_i + \tilde{\alpha}\Delta t + \sqrt{\tilde{\beta}\Delta t} z_{i+1}$$

- Sometimes want to ensure non-negativity of xs
- So we use

$$x_{i+1} = h\left(x_i + \tilde{\alpha}\Delta t + \sqrt{\tilde{\beta}\Delta t} z_{i+1}\right)$$

- Where h outputs non-negative values e.g. softplus function

$$h(z) = \log(1 + e^z)$$

Variational approximation to diffusion

- Drift and diffusion calculated from neural network
- Used to calculate x_1
- Fed back into same neural network to get next drift and diffusion
- ...
- **Recurrent neural network** structure
- Powerful but tricky to scale up

Algorithm summary

Initialise ϕ_θ, ϕ_x

Begin loop

Sample θ from $q(\theta; \phi_\theta)$ (independent normals)

Sample x from $q(x; \theta, \phi_x)$ (run RNN)

Calculate ELBO gradient

$$\nabla \hat{\mathcal{L}}(\phi) = \nabla_\phi \log \frac{p(\theta, x, y)}{q(\theta; \phi_\theta) q(x|\theta; \phi_x)}$$

Update ϕ_θ, ϕ_x by stochastic optimisation

End loop

(n.b. can use larger Monte Carlo batch size)

Algorithm summary

Initialise ϕ_θ, ϕ_x

Begin loop

Sample θ from $q(\theta; \phi_\theta)$ (independent normals)

Sample x from $q(x; \theta, \phi_x)$ (run RNN)

Calculate ELBO gradient

$$\nabla \hat{\mathcal{L}}(\phi) = \nabla_\phi \log \frac{p(\theta, x, y)}{q(\theta; \phi_\theta) q(x|\theta; \phi_x)}$$

Update ϕ_θ, ϕ_x by stochastic optimisation

End loop

(n.b. can use larger Monte Carlo batch size)

Example

Lotka-Volterra example

- Classic population dynamics model
- Two populations: prey and predators
- Three processes
  - Prey growth
  - Predator growth (by consuming prey)
  - Predator death
- Many variations exist
- We use SDE from Golightly and Wilkinson (2011)

Lotka-Volterra example

- Prey population at time t is U_t
- Predator population at time t is V_t
- Drift

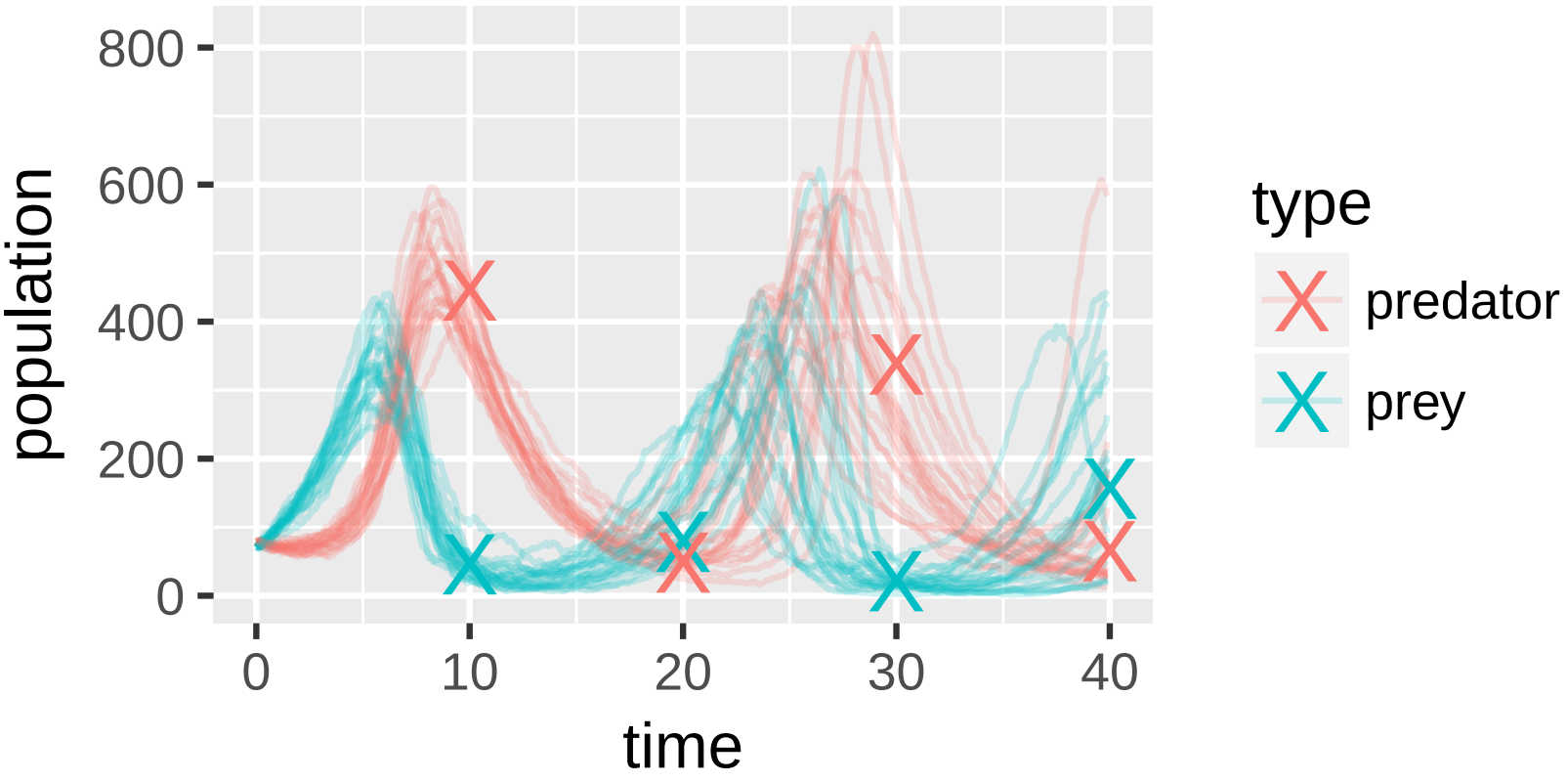
$$\alpha(X_t, \theta) = \begin{pmatrix} \theta_1 U_t - \theta_2 U_t V_t \\ \theta_2 U_t V_t - \theta_3 V_t \end{pmatrix}$$

- Diffusion

$$\beta(X_t, \theta) = \begin{pmatrix} \theta_1 U_t + \theta_2 U_t V_t & -\theta_2 U_t V_t \\ -\theta_2 U_t V_t & \theta_3 V_t + \theta_2 U_t V_t \end{pmatrix}$$

- Parameters:
  - θ_1 controls prey growth
  - θ_2 controls predator growth by consuming prey
  - θ_3 controls predator death

Lotka-Volterra example



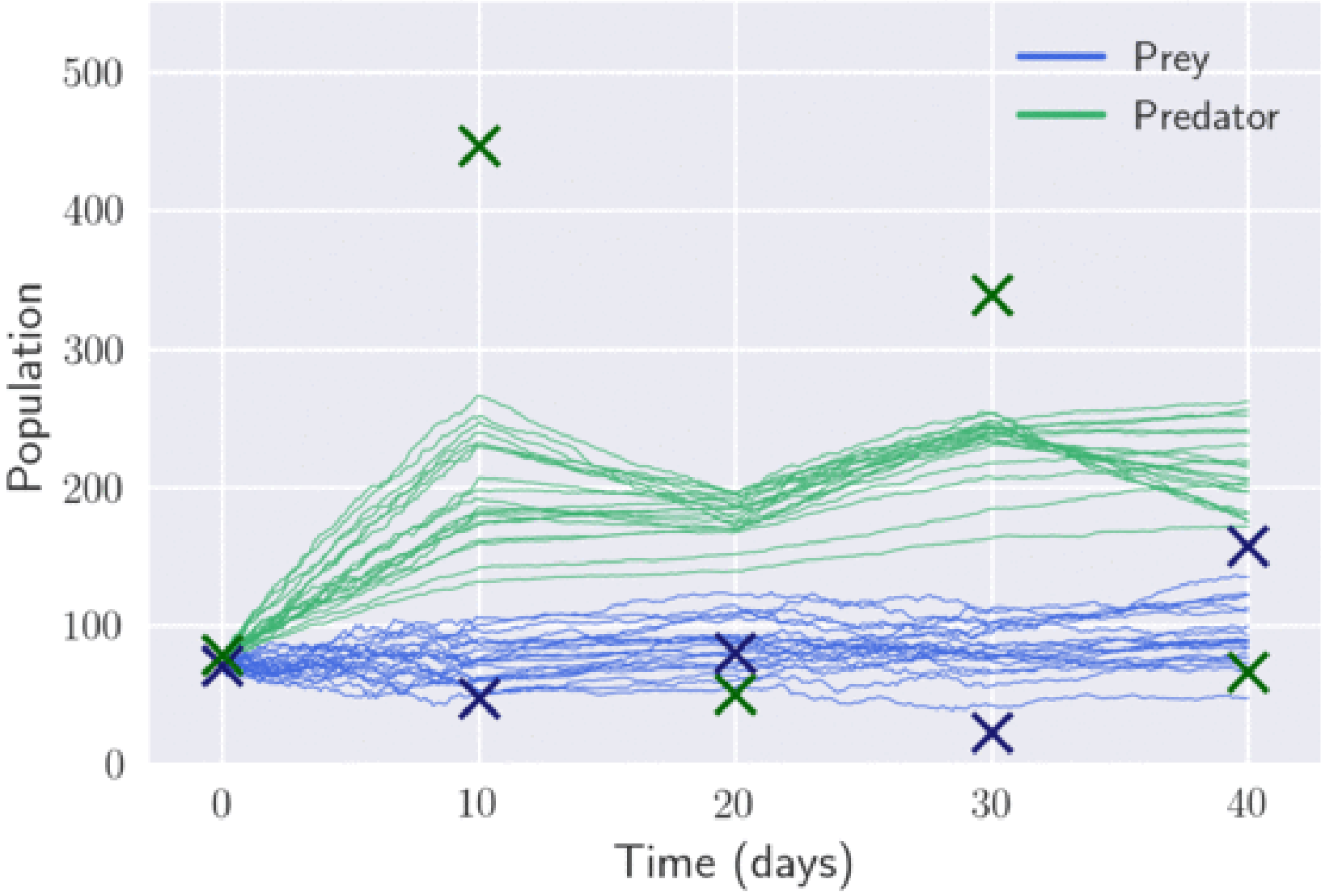
Settings - model

- IID priors: $\log \theta_i \sim N(0, 3^2)$ for $i = 1, 2, 3$
- Discretisation time step $\Delta\tau = 0.1$
- Observation variance $\Sigma = \mathrm{I}$ - small relative to typical population sizes
- Challenging scenario:
  - Non-linear diffusion paths
  - Small observation variance
  - Long gaps between observations

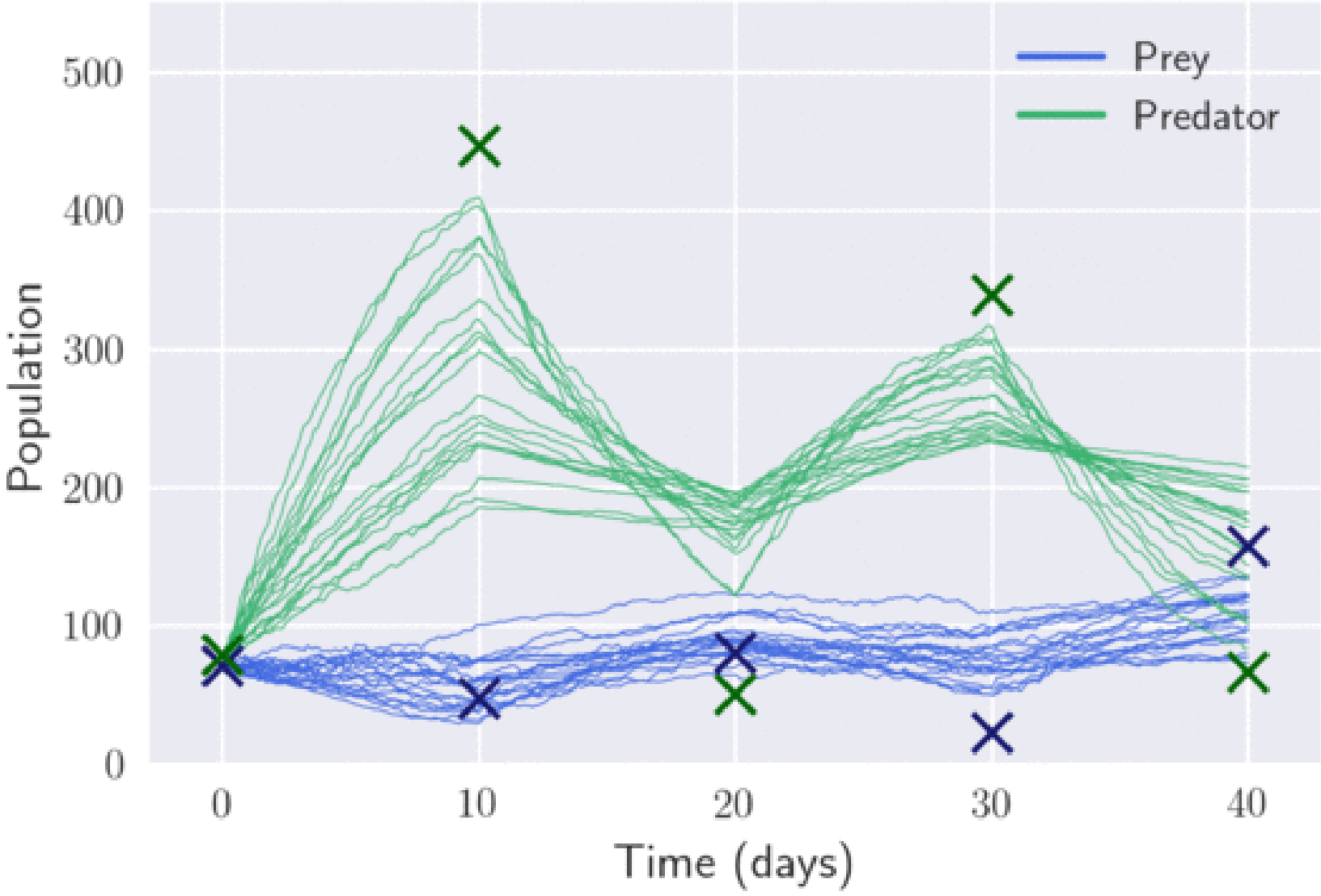
Settings - variational inference

- Batch size 50 for gradient estimate
- 4 layer neural network (20 ReLU units / layer)
- Softplus transformation to avoid proposing negative population levels
- Various methods to avoid numerical problems in training

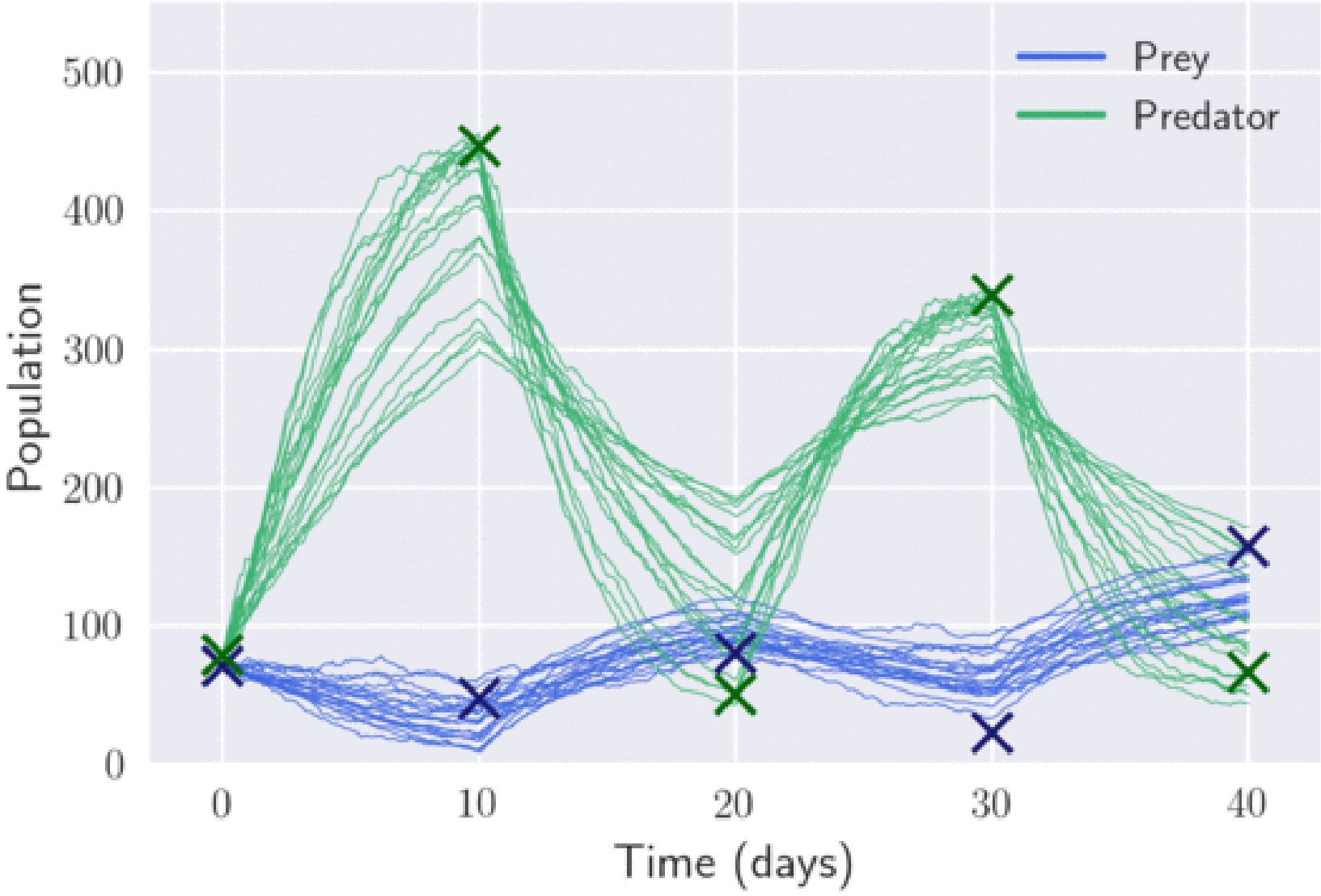
Results


Prey
Predator
Population
500
400
300
200
100
0
0
10
20
30
40
Time (days)

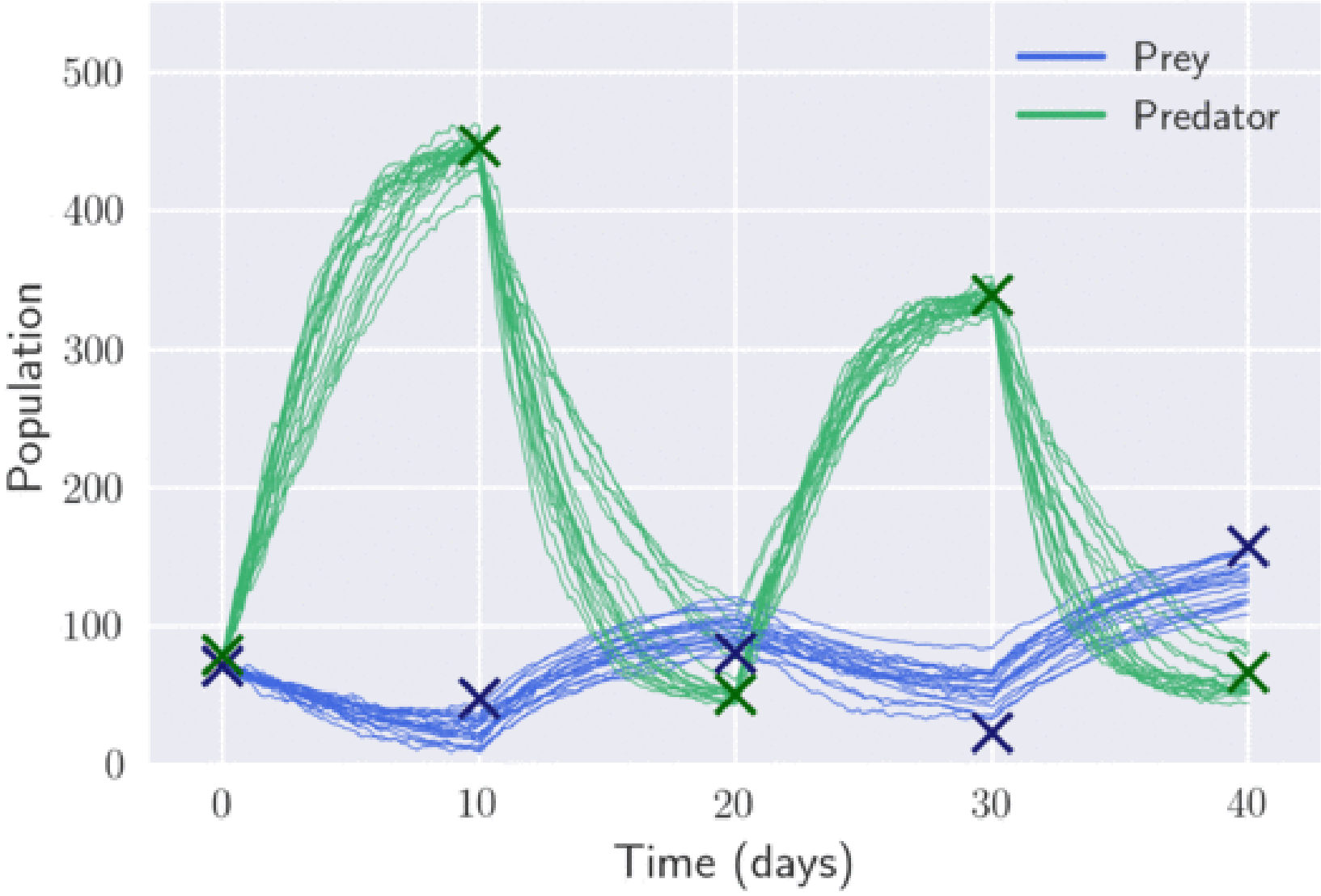
Results


Prey
Predator
Population
500
400
300
200
100
0
0
10
20
30
40
Time (days)

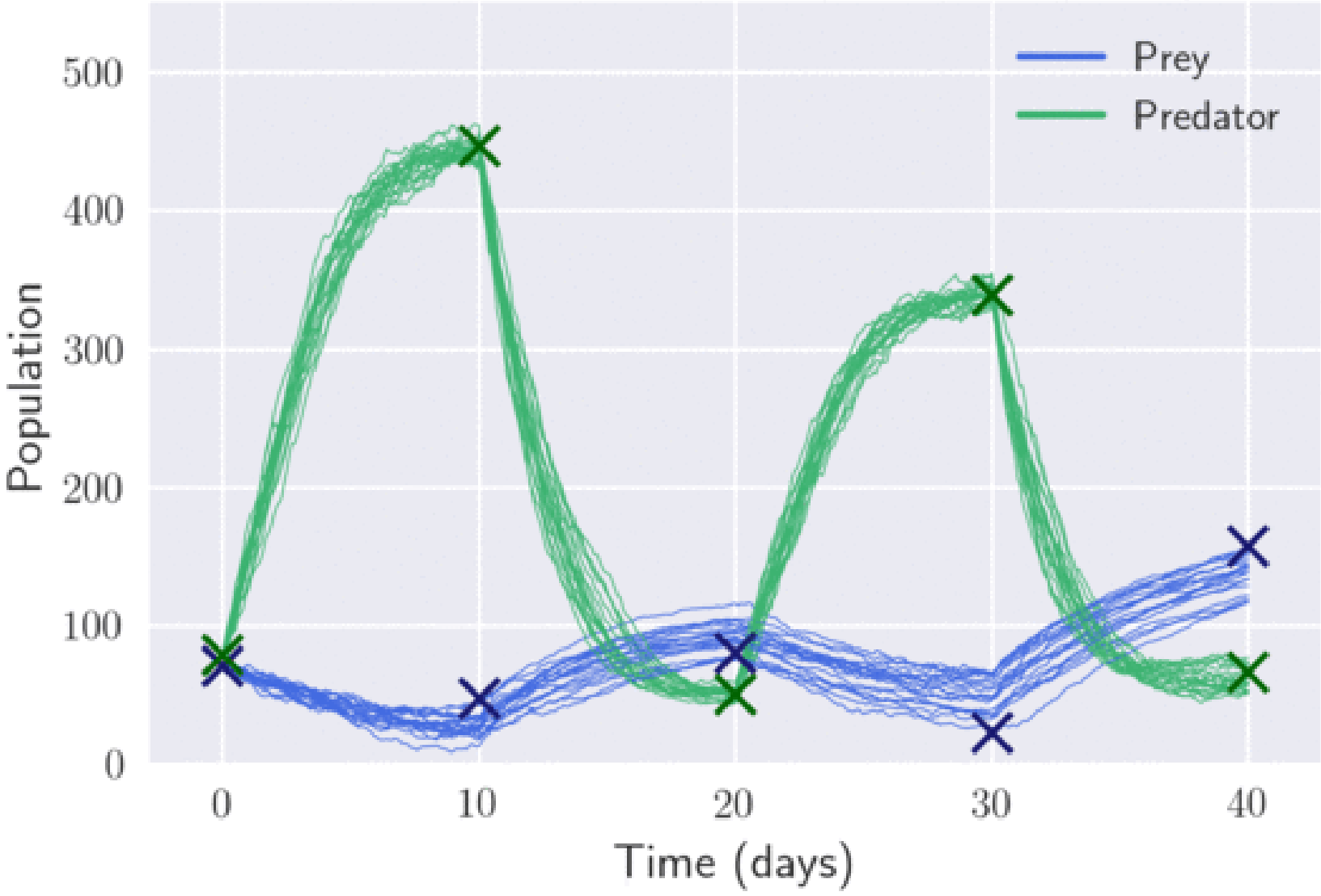
Results


Prey
Predator
Population
500
400
300
200
100
0
0
10
20
30
40
Time (days)

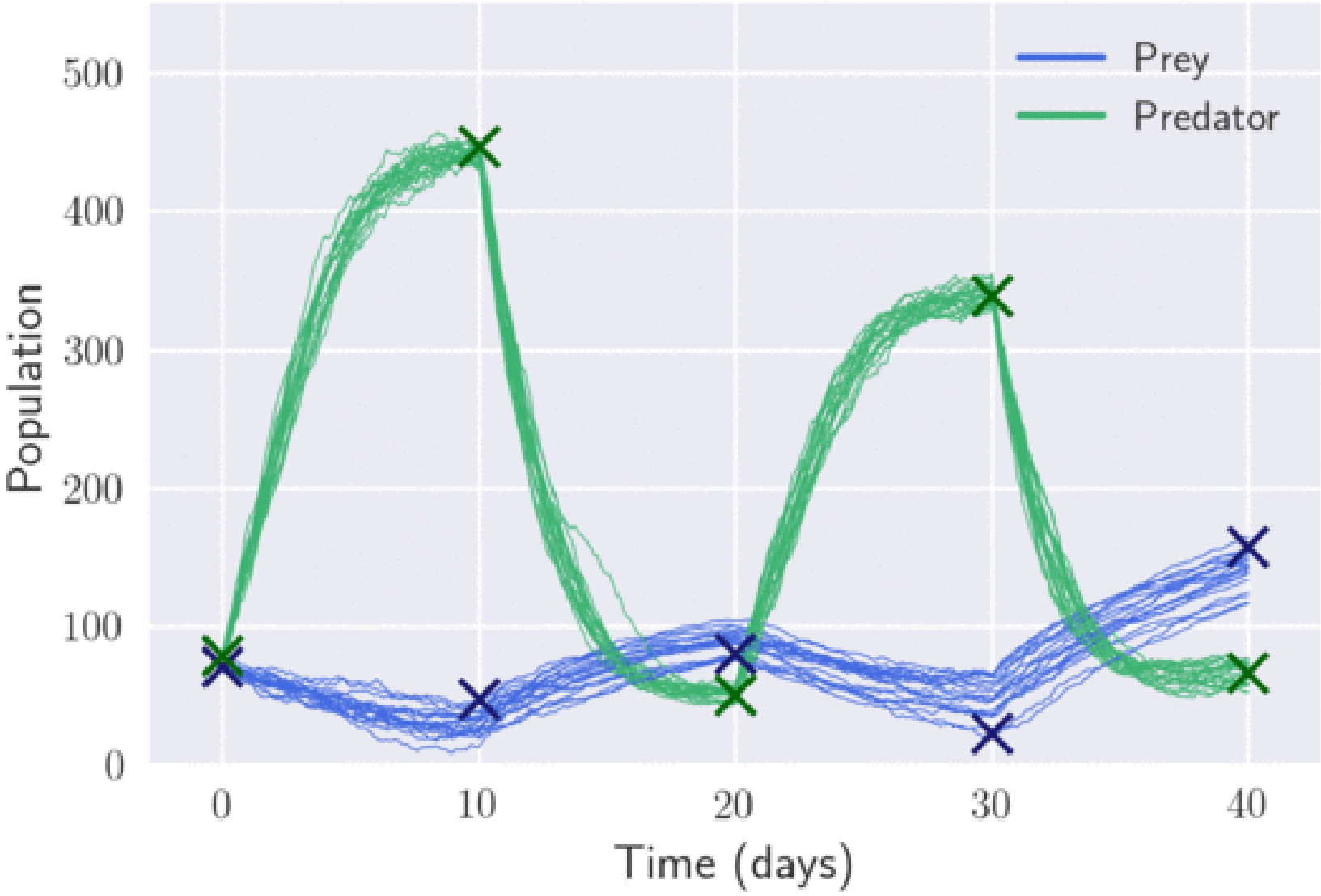
Results


Prey
Predator
Population
500
400
300
200
100
0
0
10
20
30
40
Time (days)

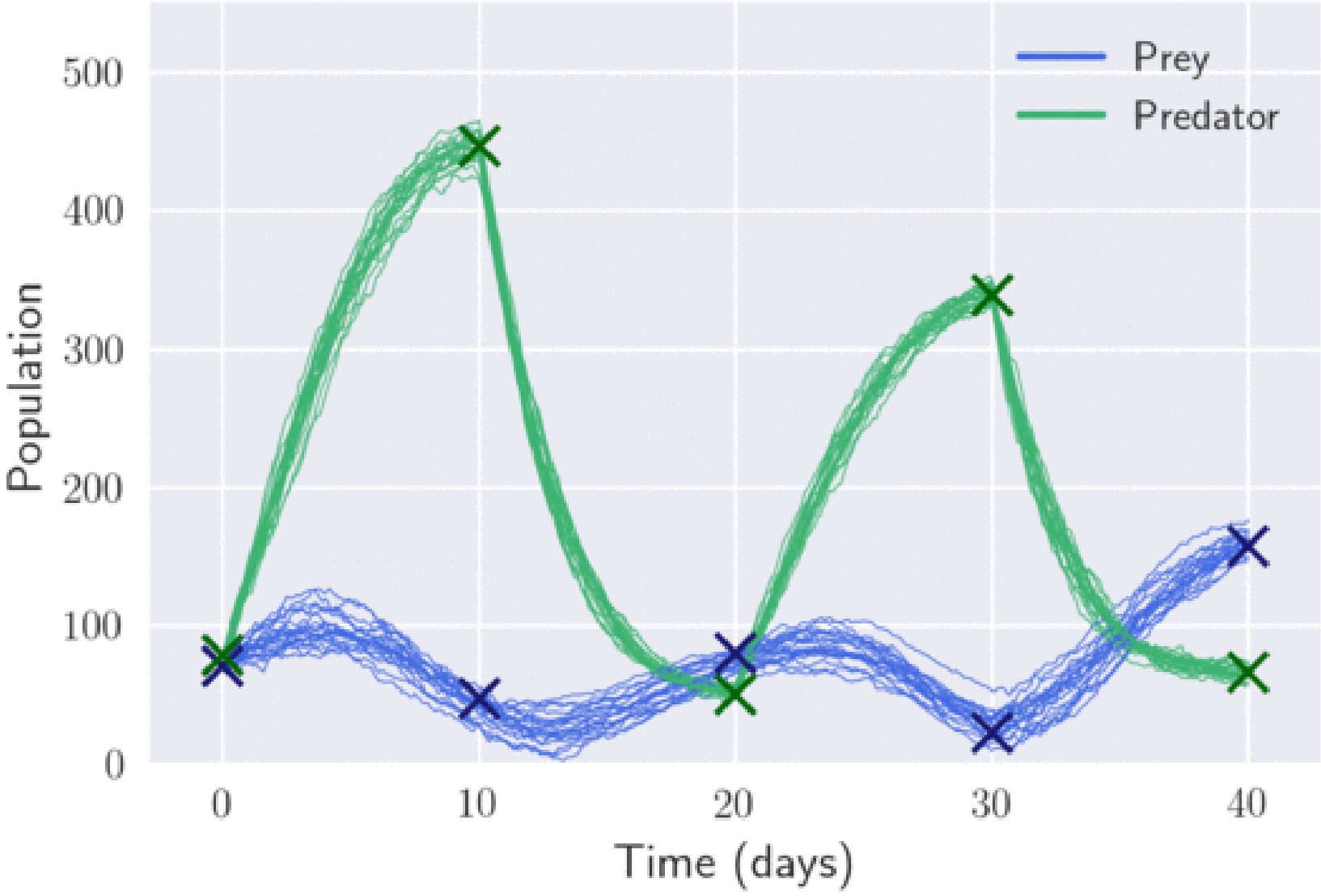
Results


Prey
Predator
Population
500
400
300
200
100
0
0
10
20
30
40
Time (days)

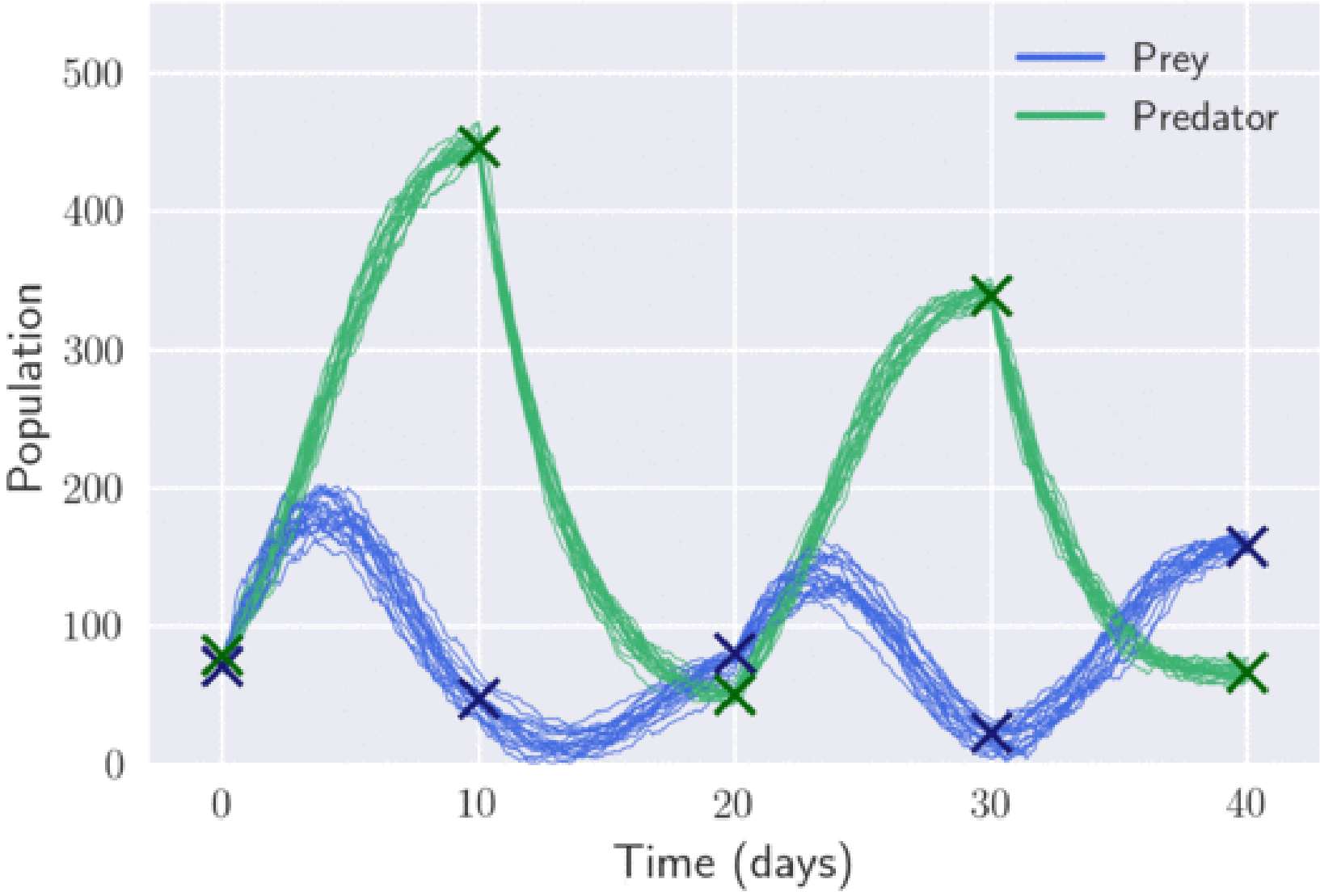
Results


Prey
Predator
Population
500
400
300
200
100
0
0
10
20
30
40
Time (days)

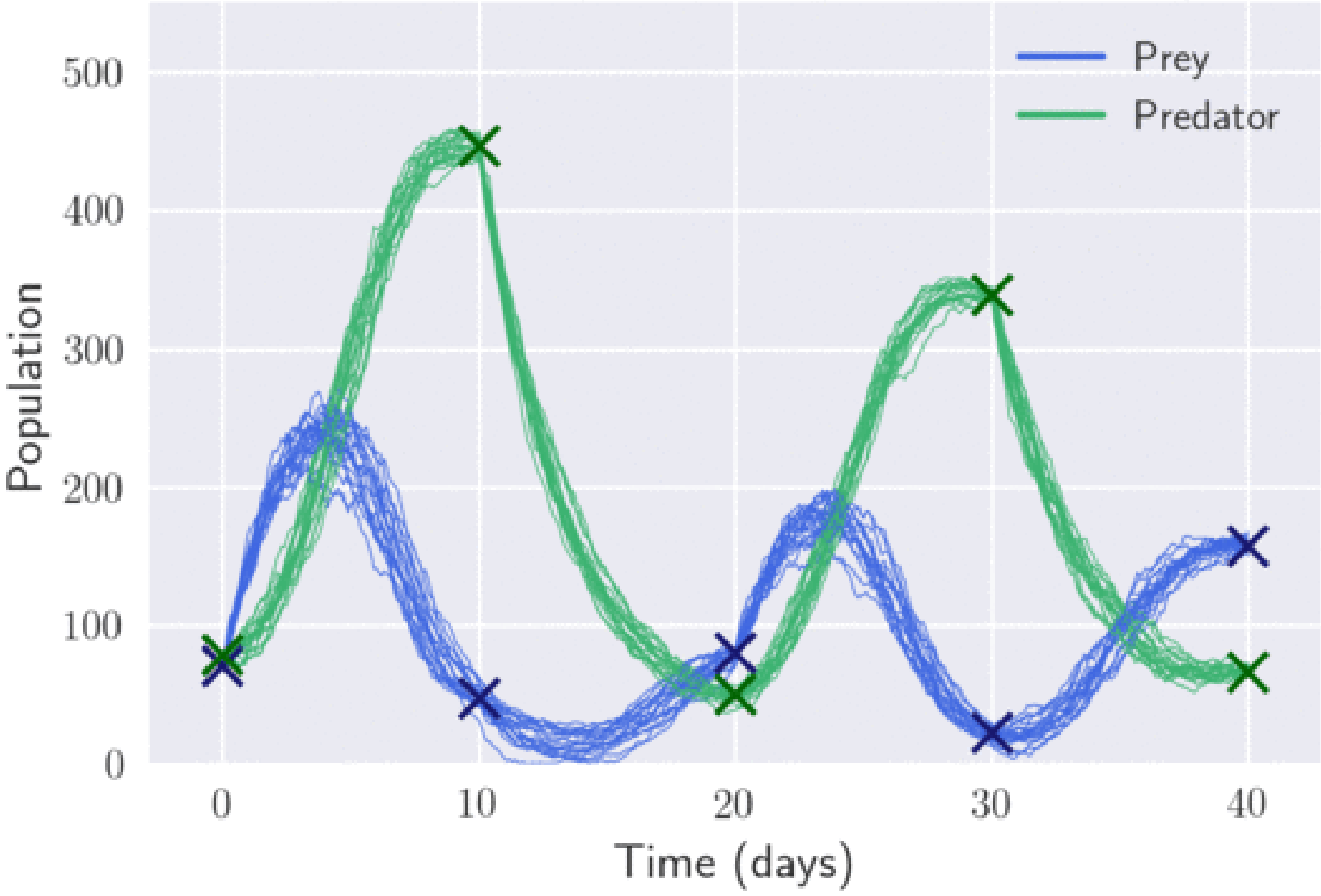
Results


Prey
Predator
Population
500
400
300
200
100
0
0
10
20
30
40
Time (days)

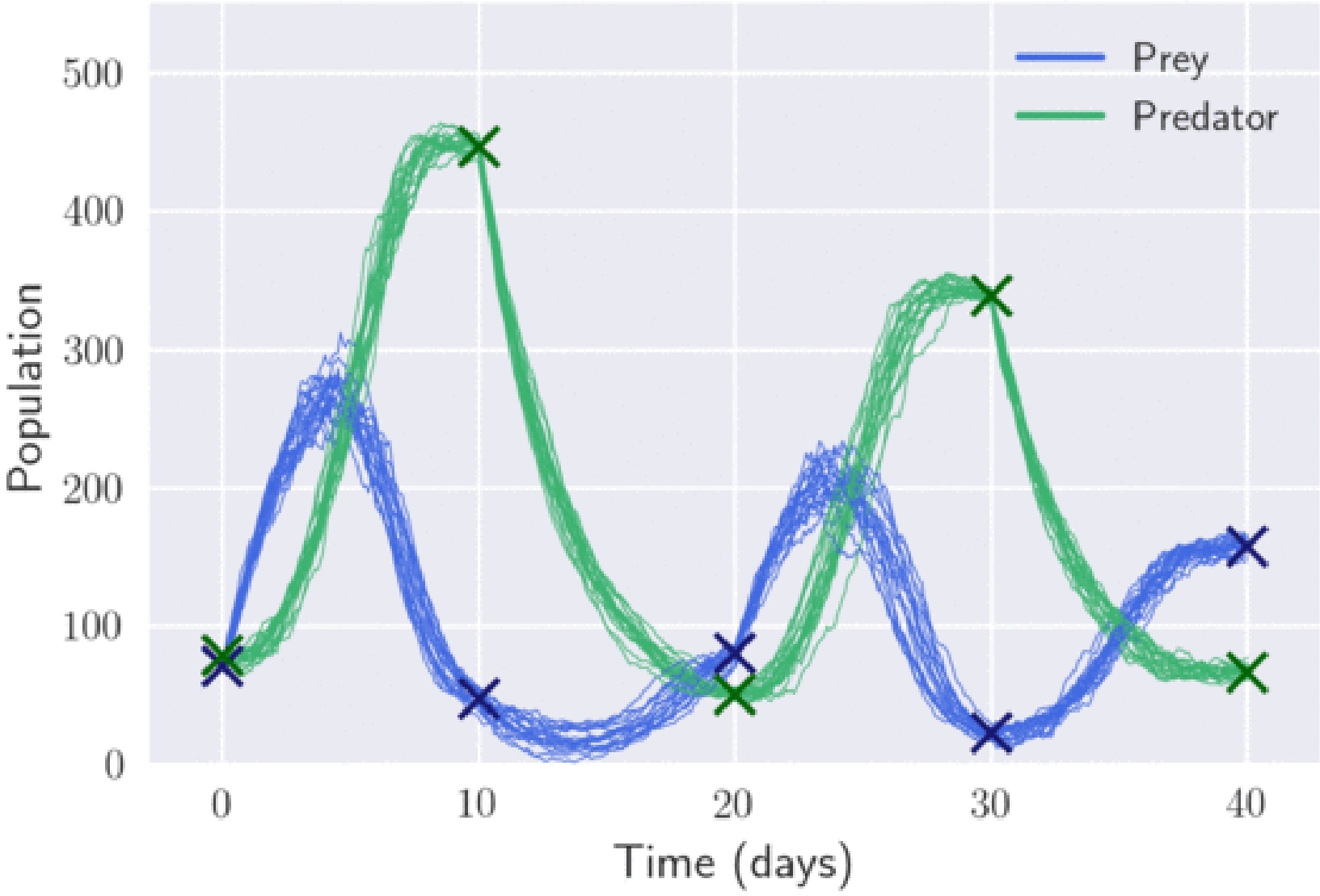
Results


Prey
Predator
Population
500
400
300
200
100
0
0
10
20
30
40
Time (days)

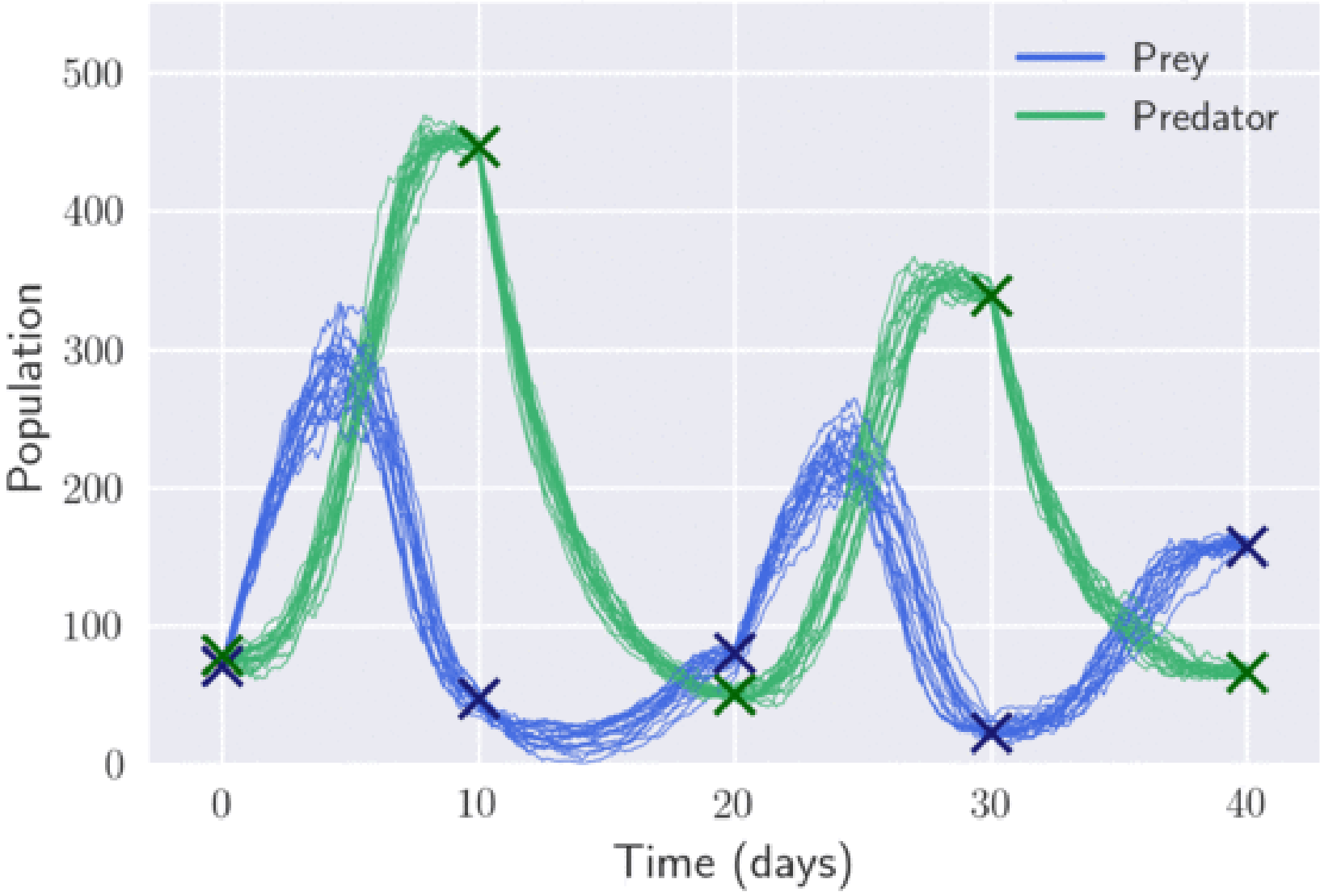
Results


Prey
Predator
Population
500
400
300
200
100
0
0
10
20
30
40
Time (days)

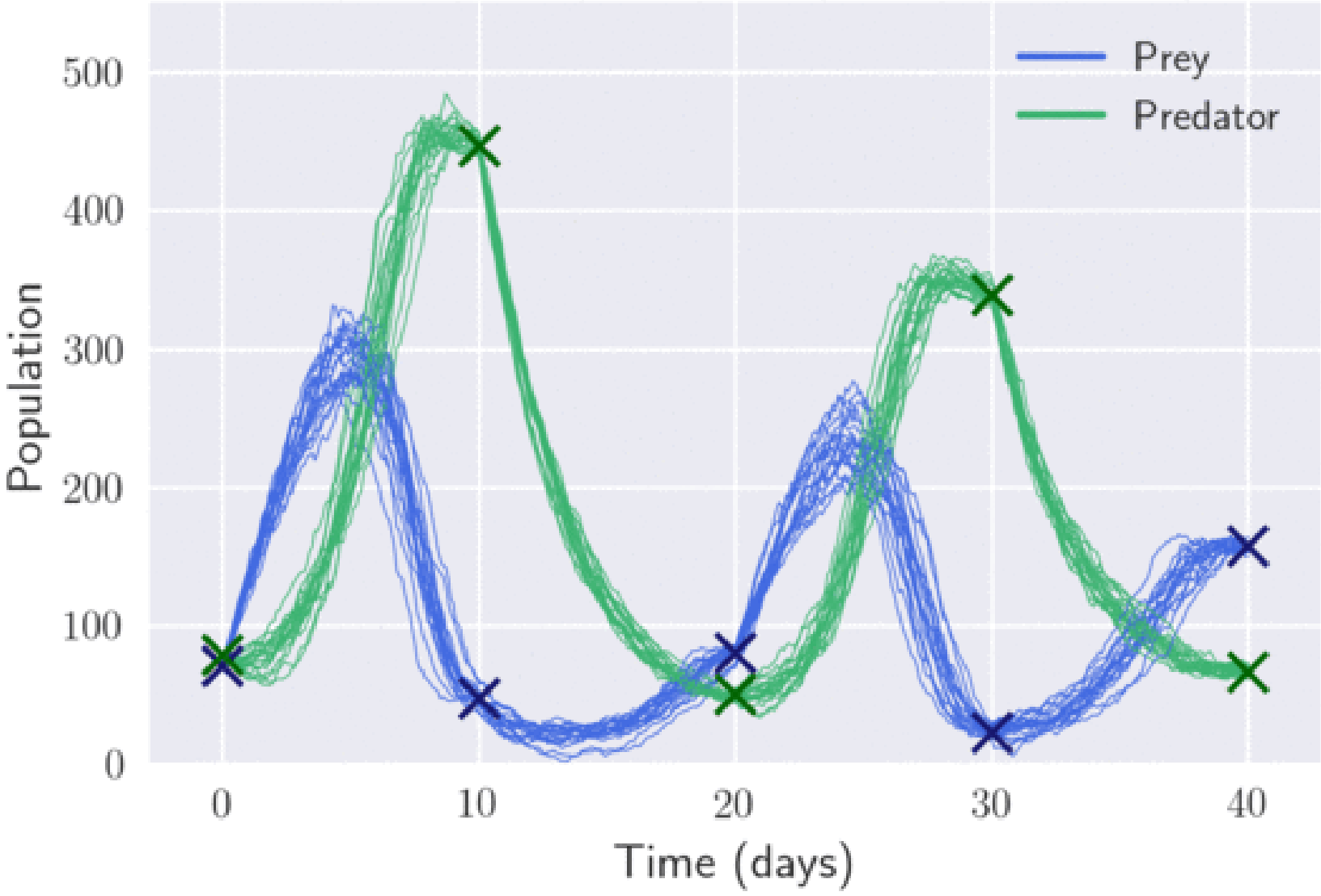
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

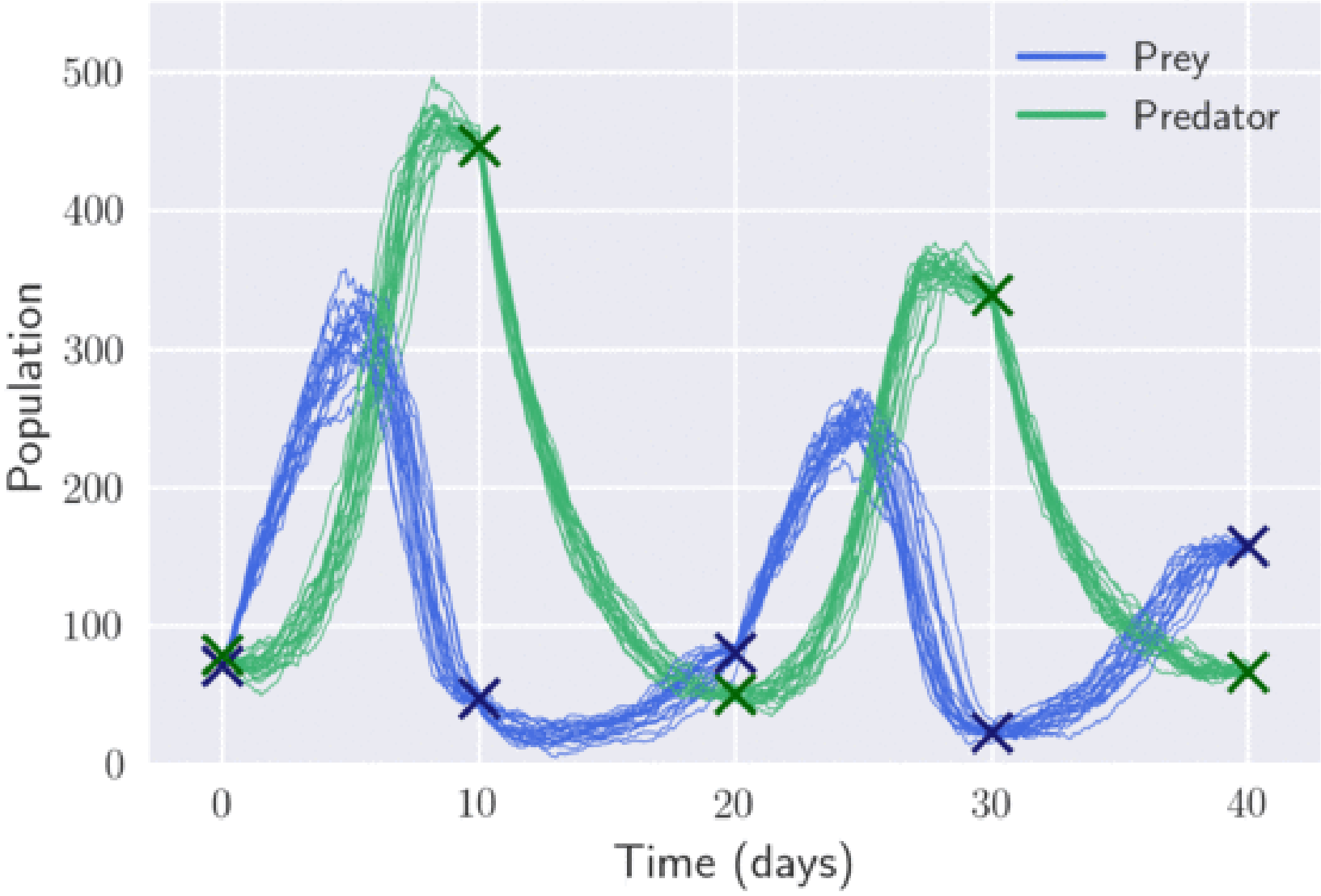
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

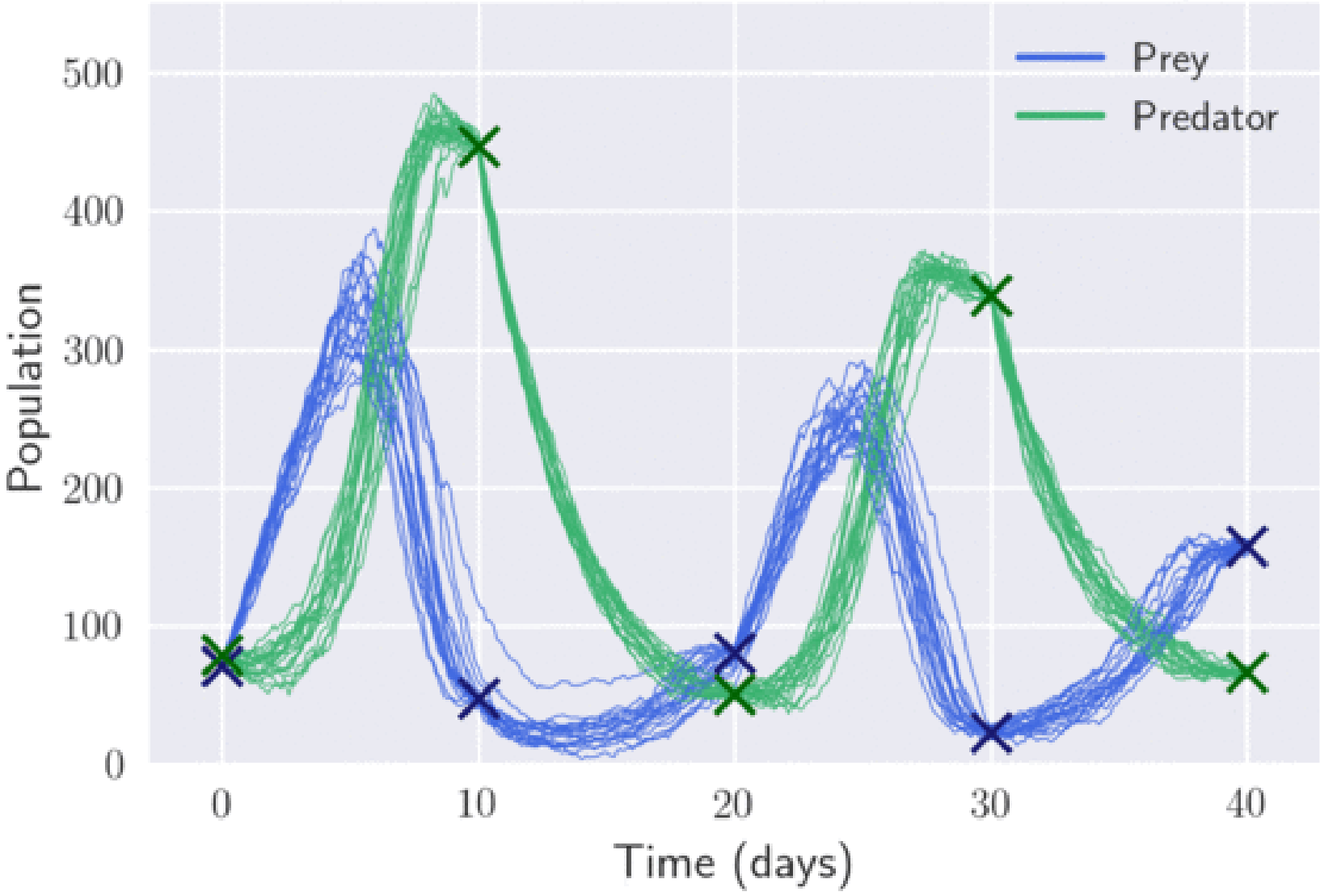
Results


Prey
Predator
Population
500
400
300
200
100
0
0
10
20
30
40
Time (days)

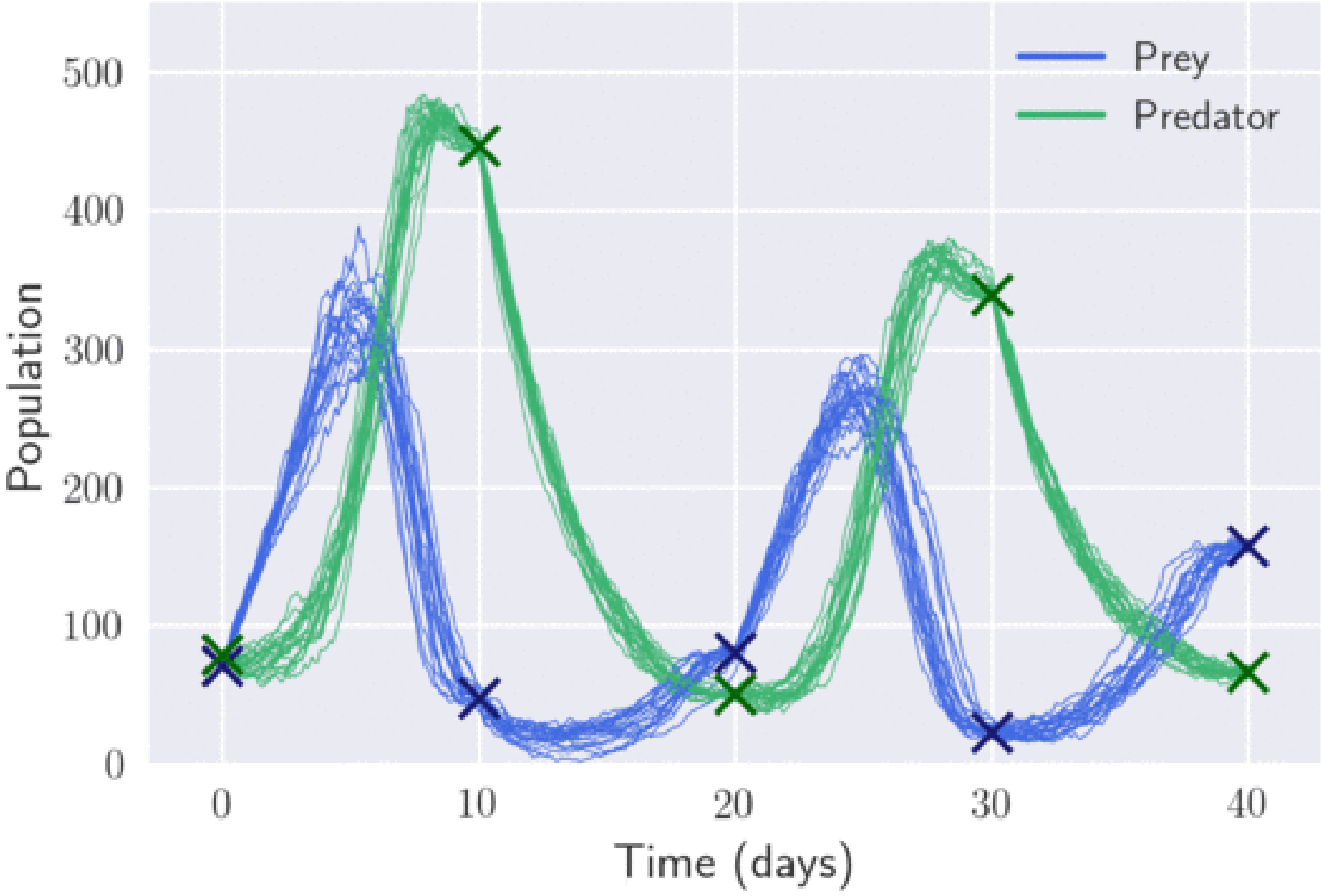
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

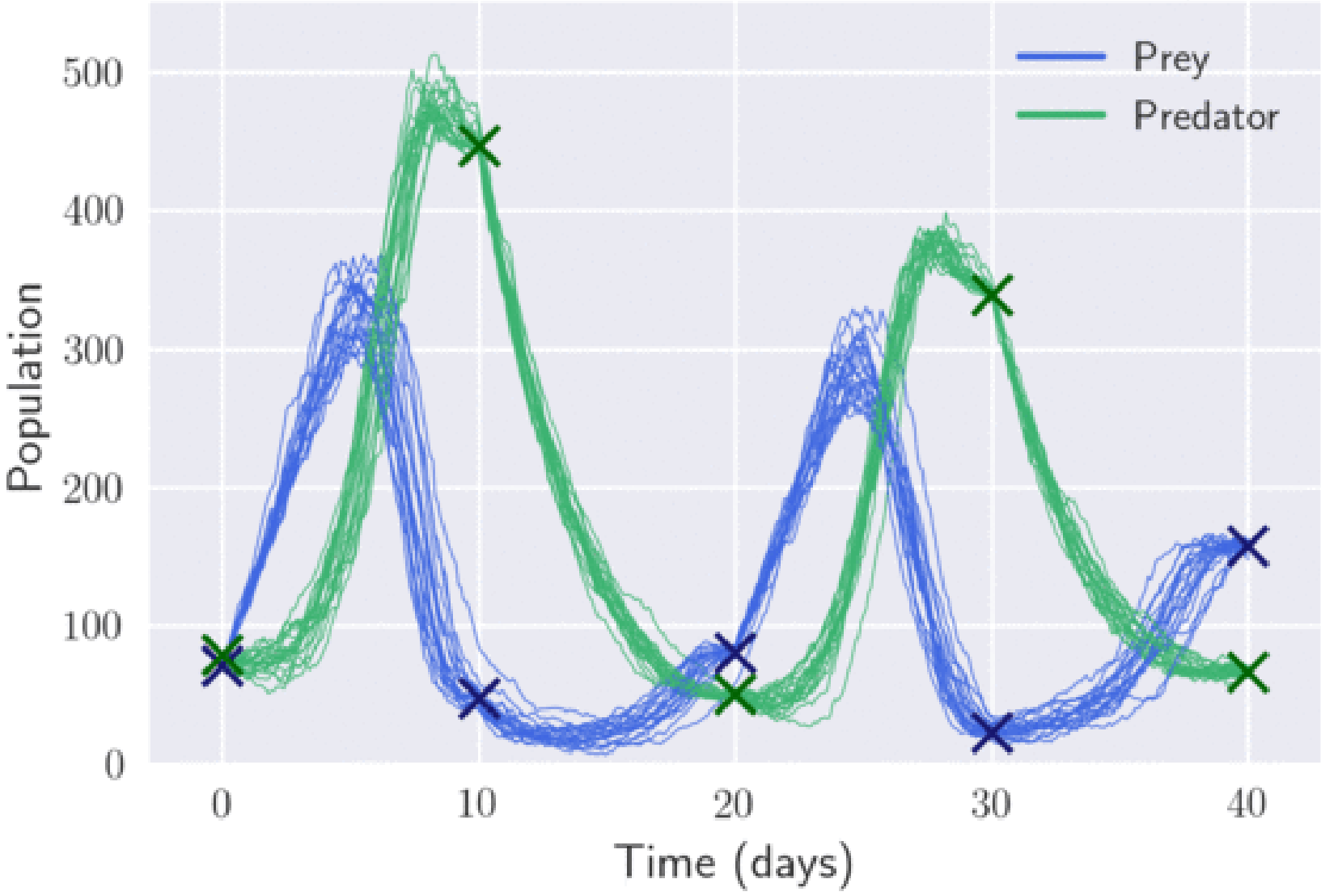
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

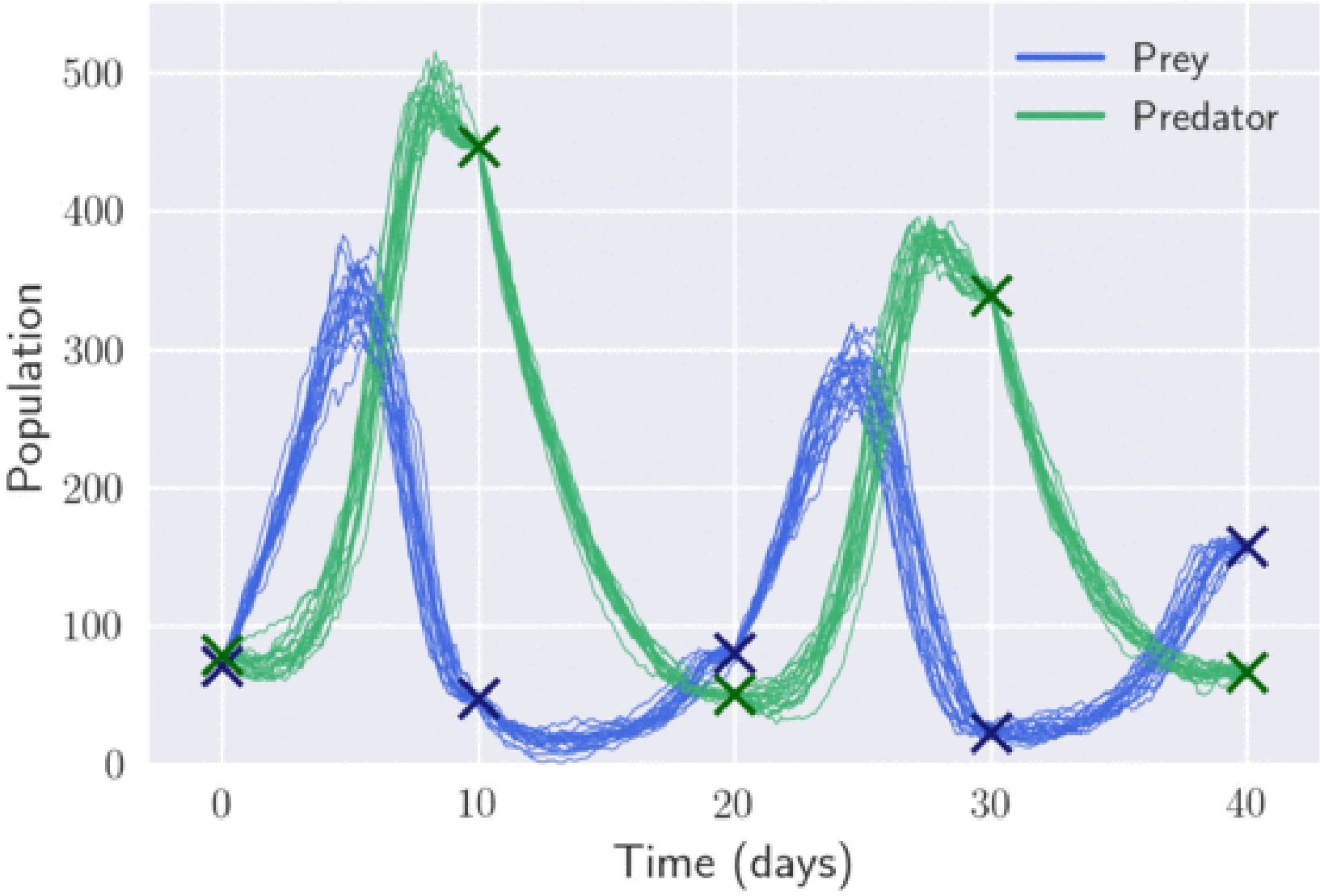
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

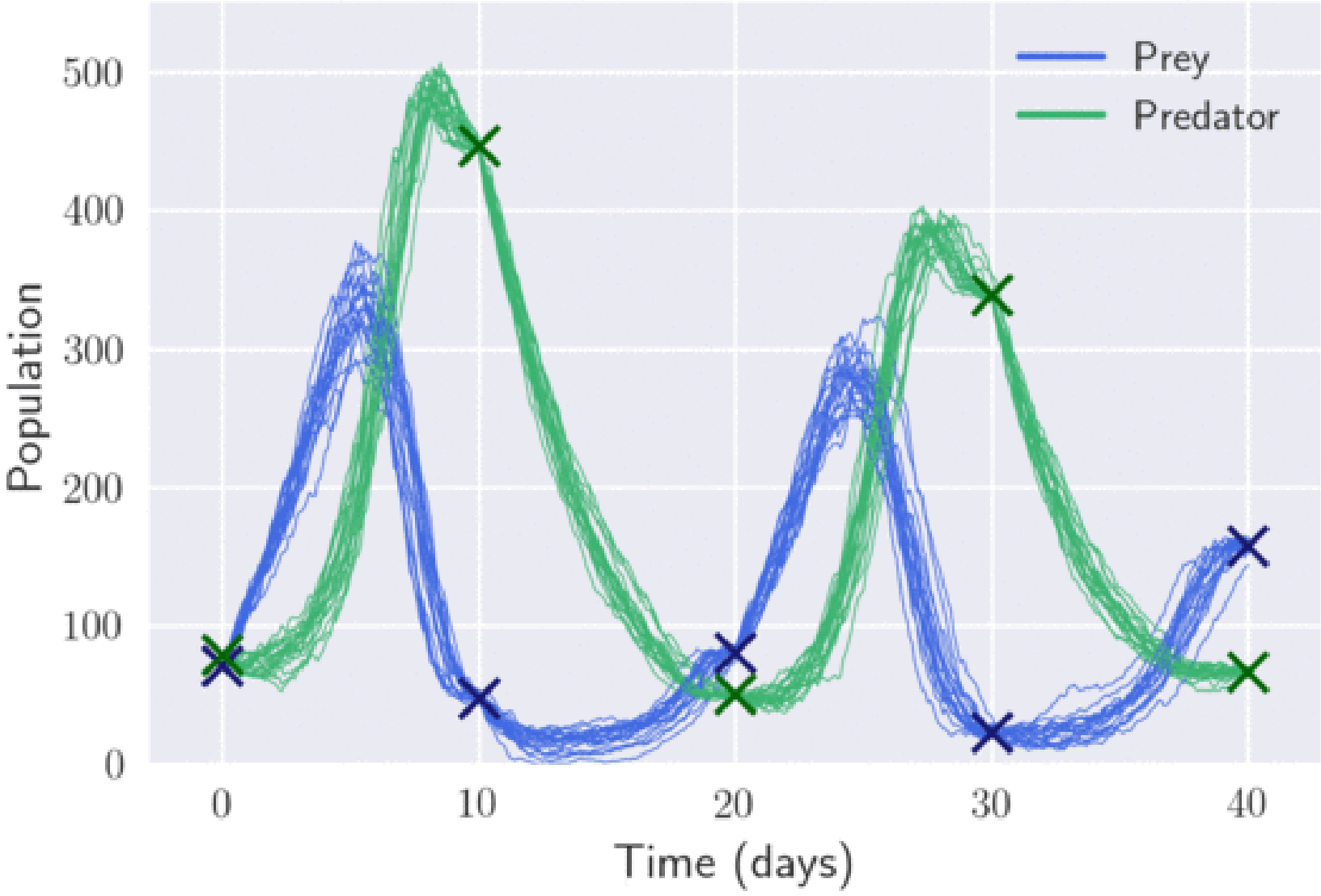
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

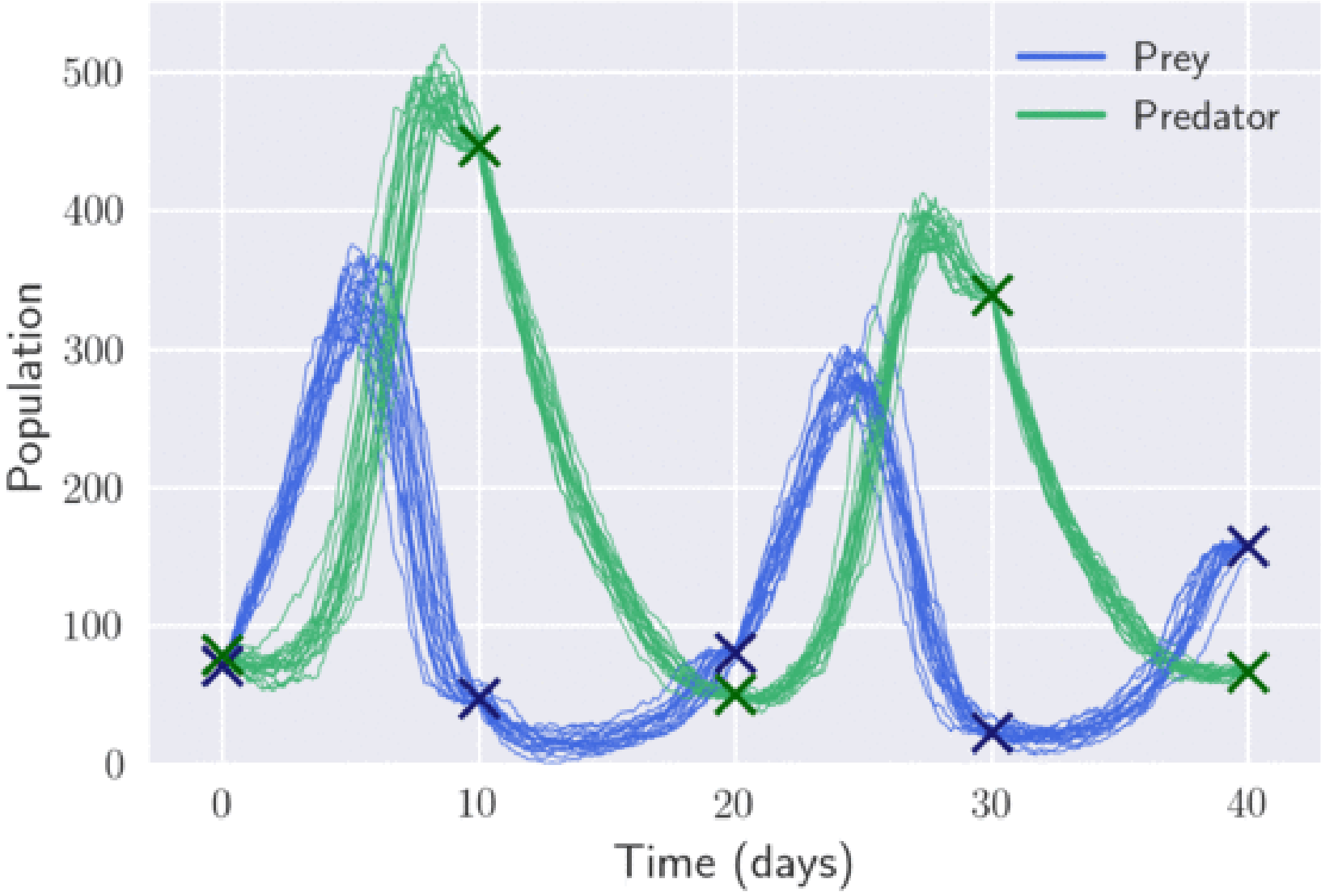
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

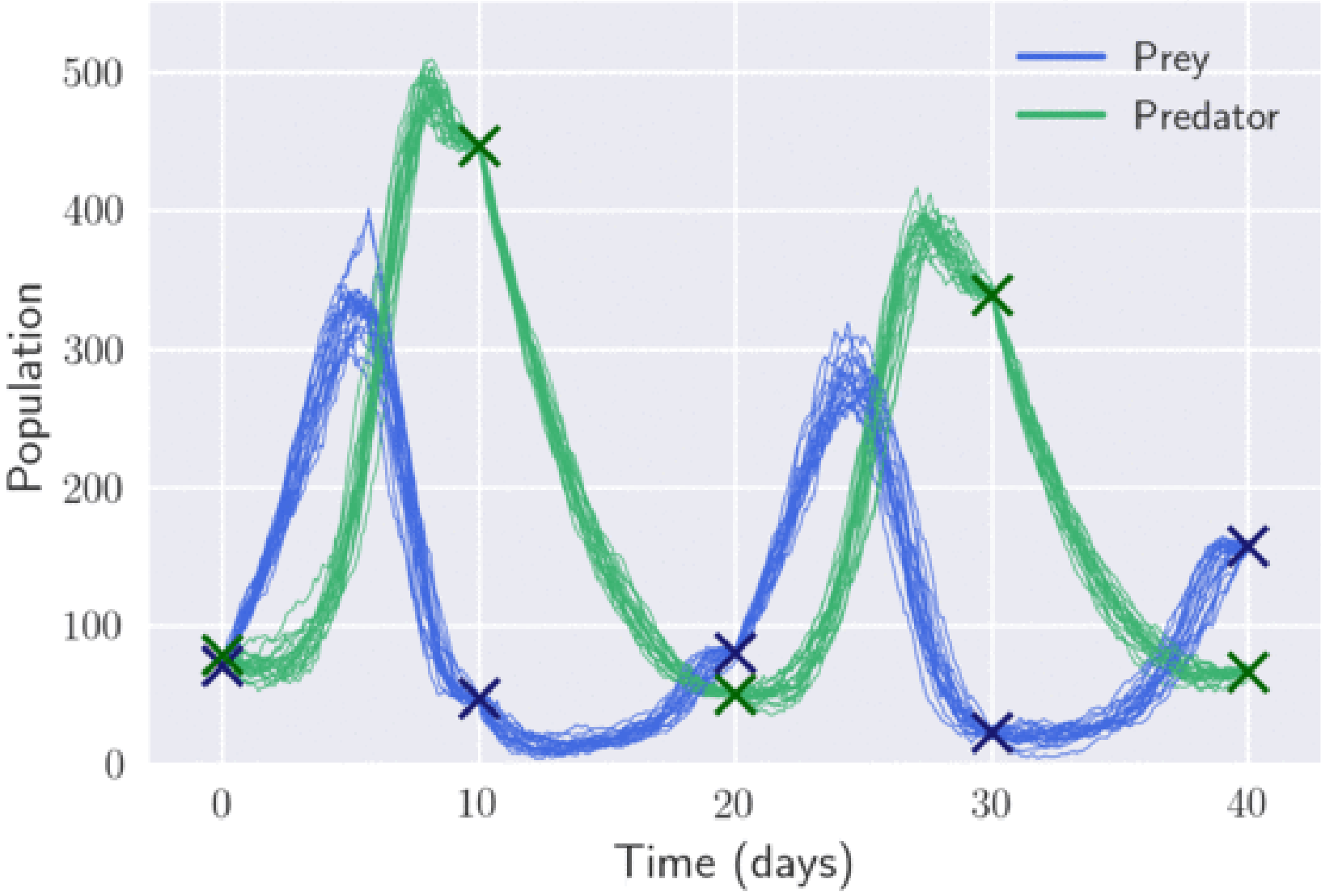
Results


Prey
Predator
Population
500
400
300
200
100
0
0
10
20
30
40
Time (days)

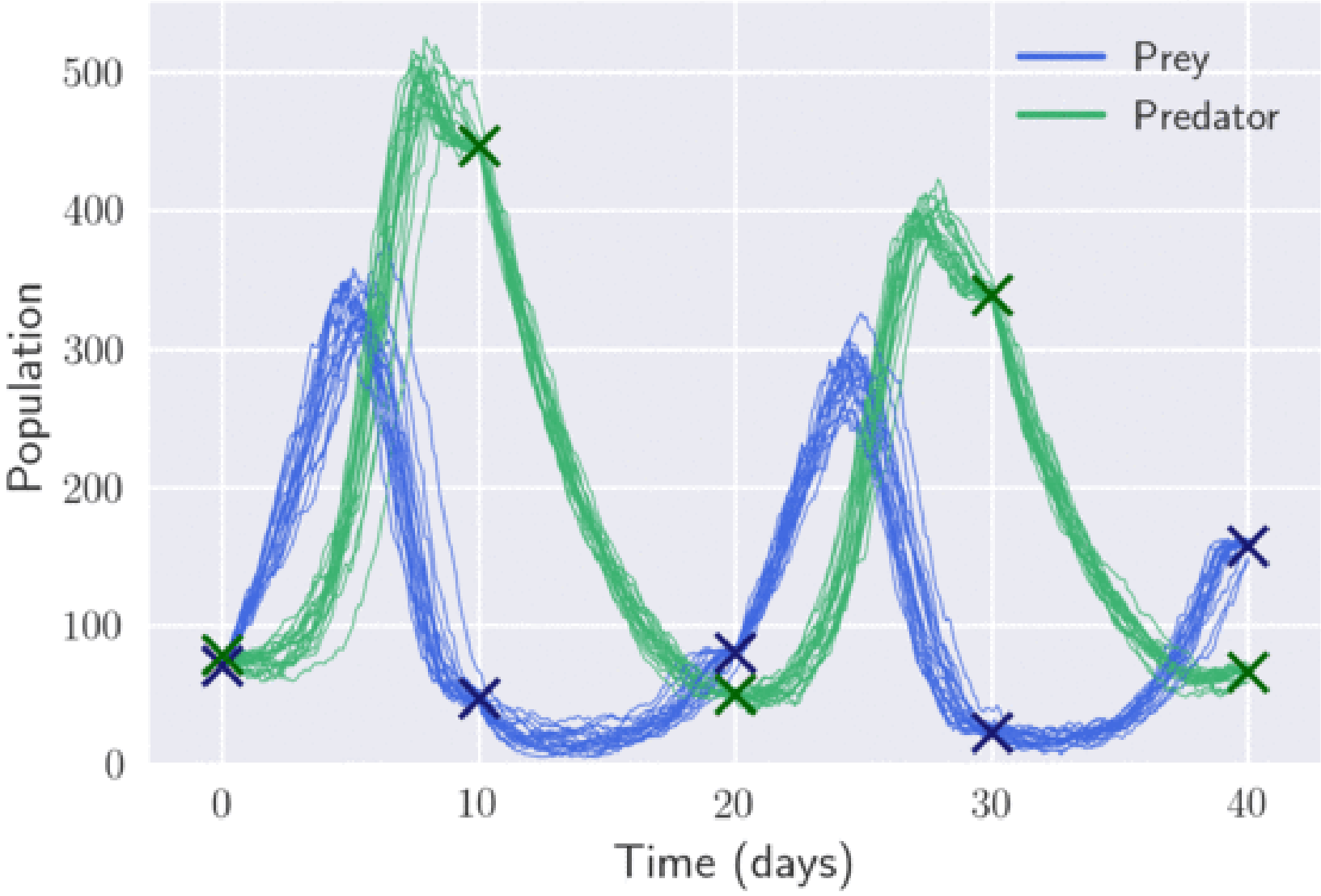
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

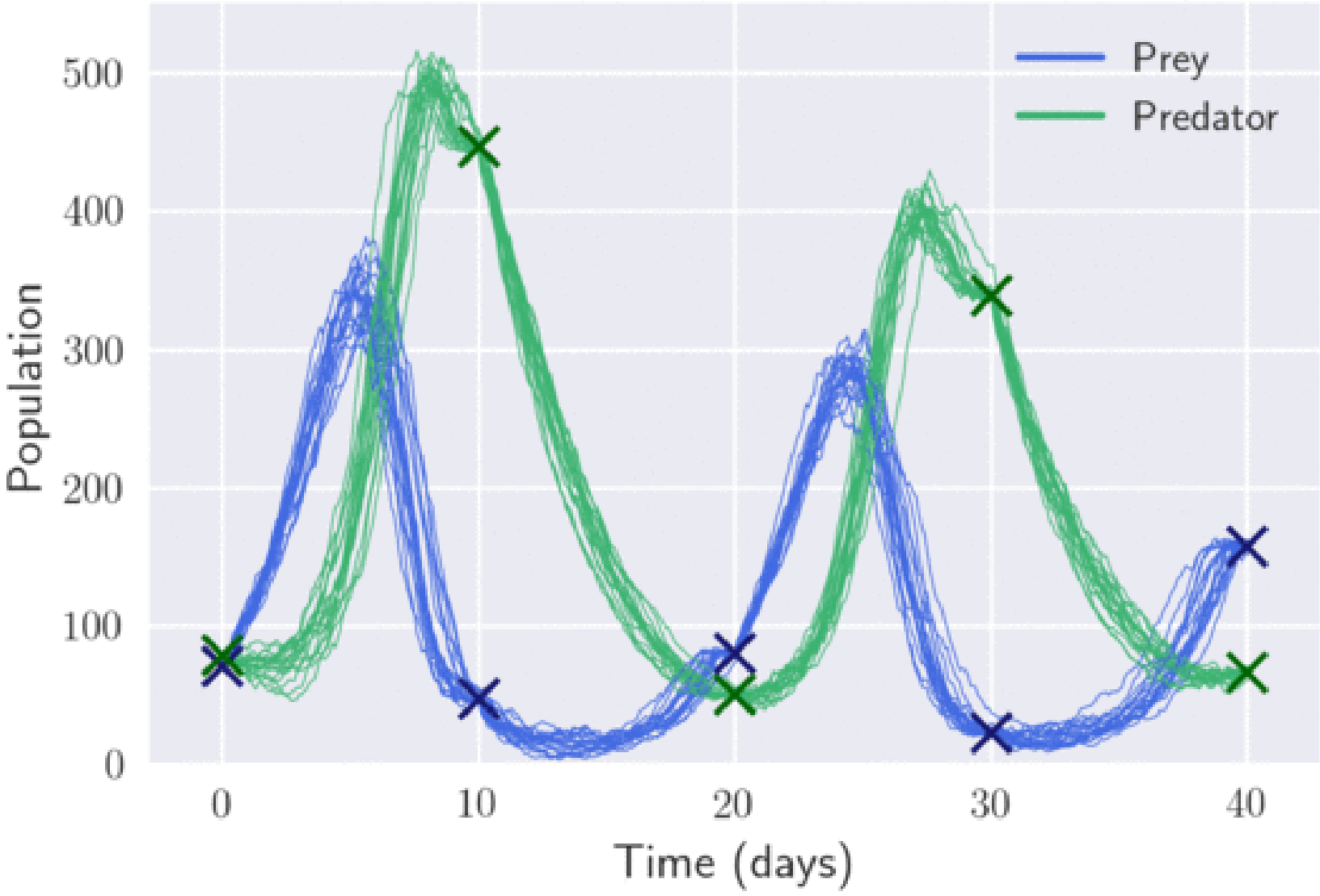
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

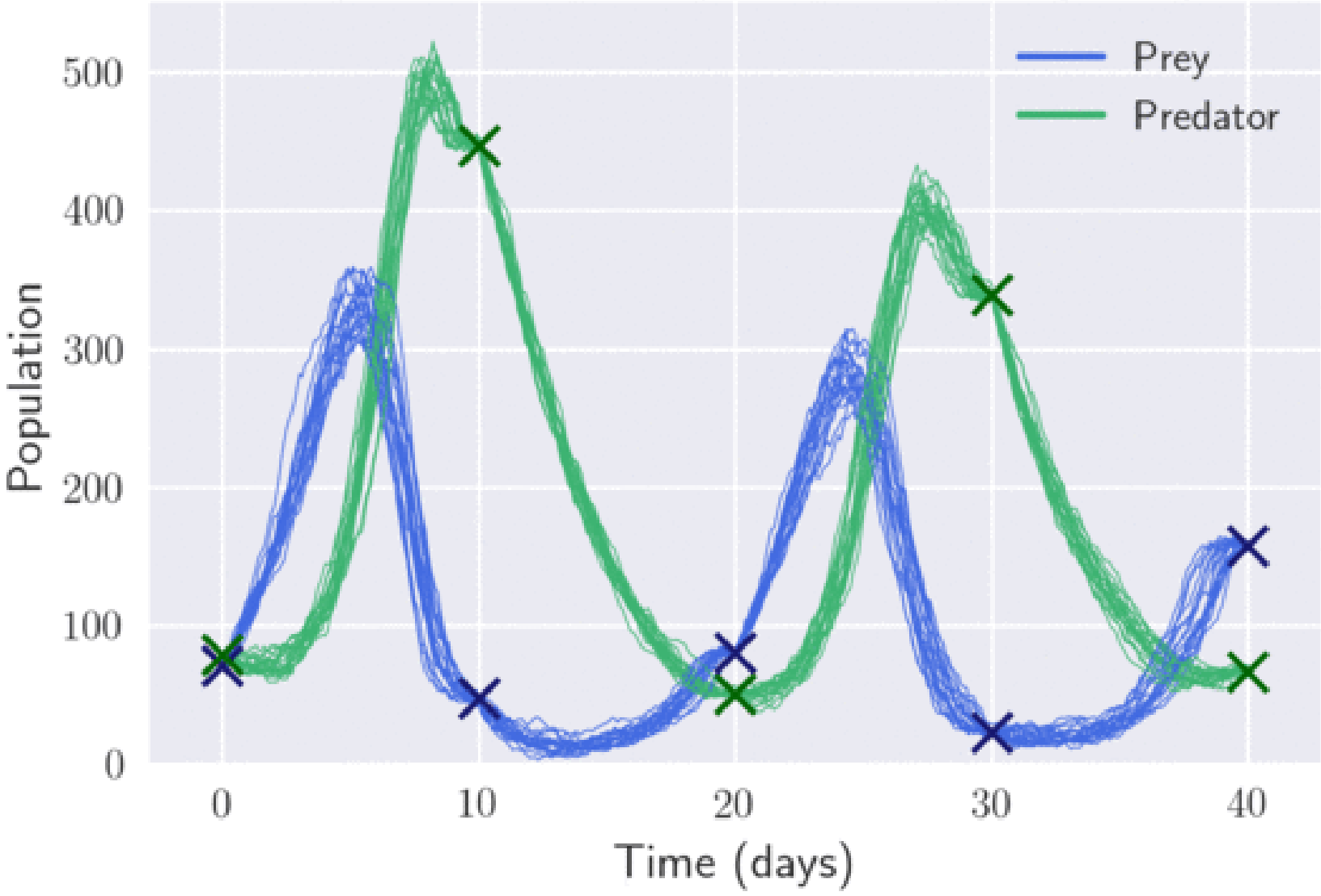
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

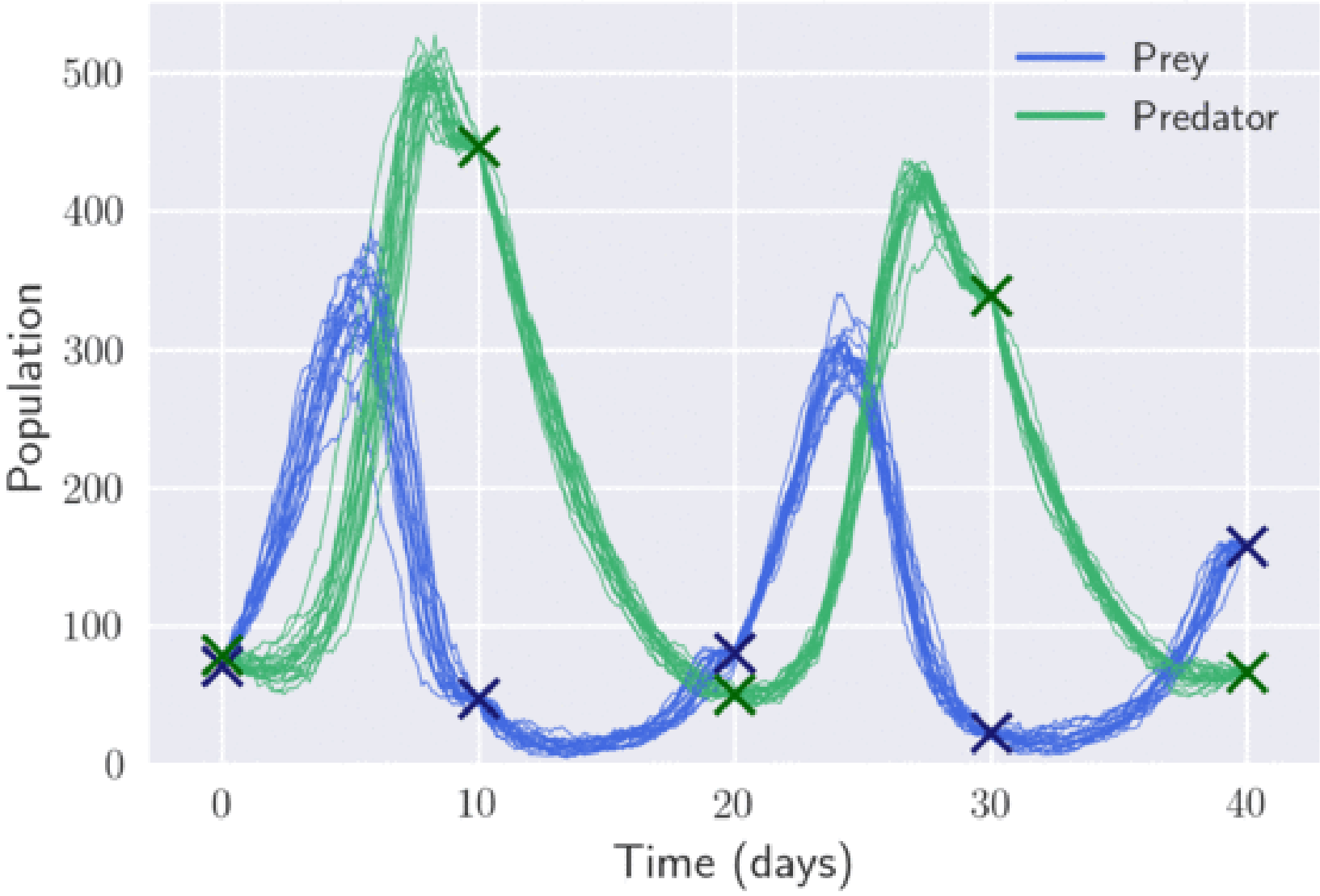
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

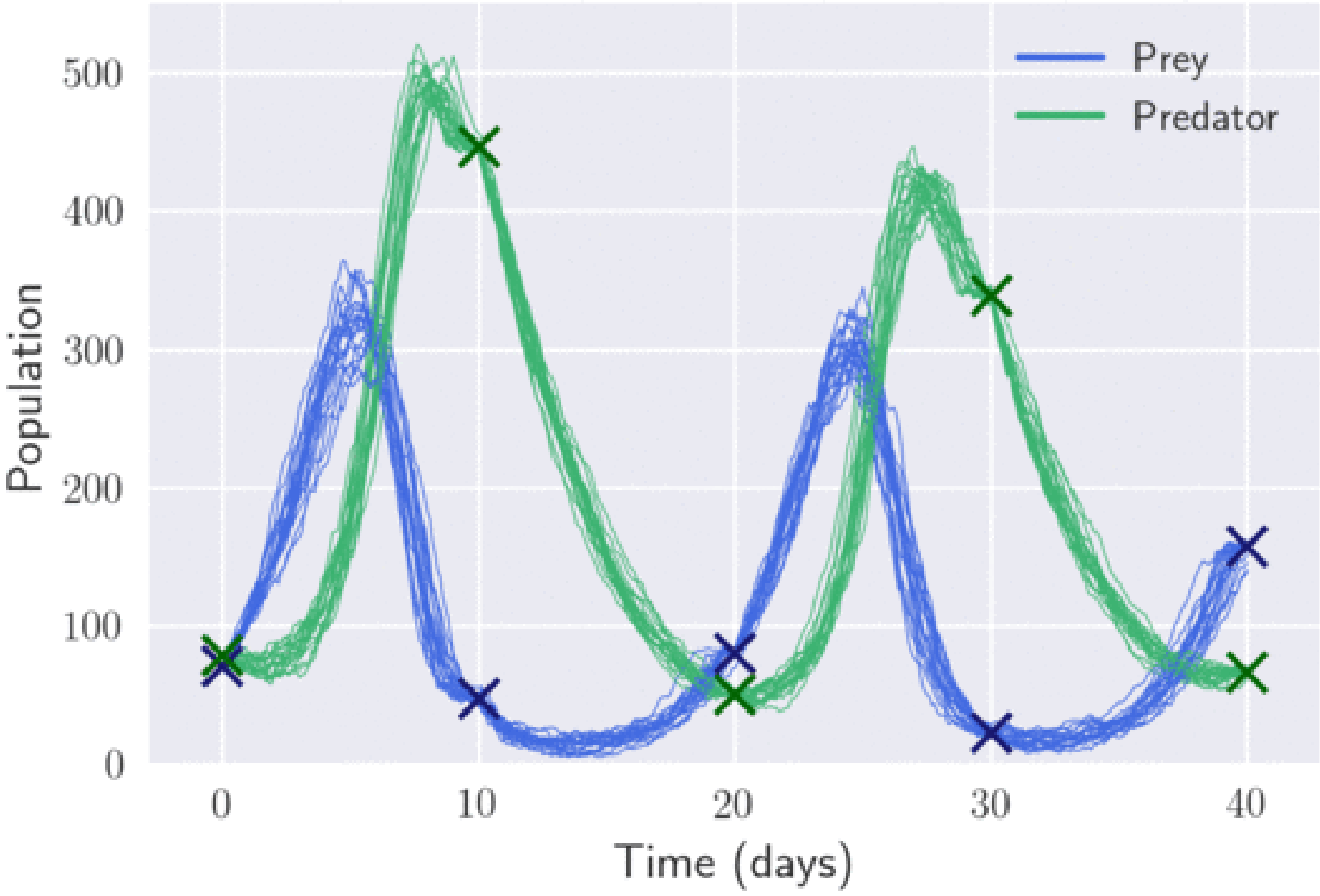
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

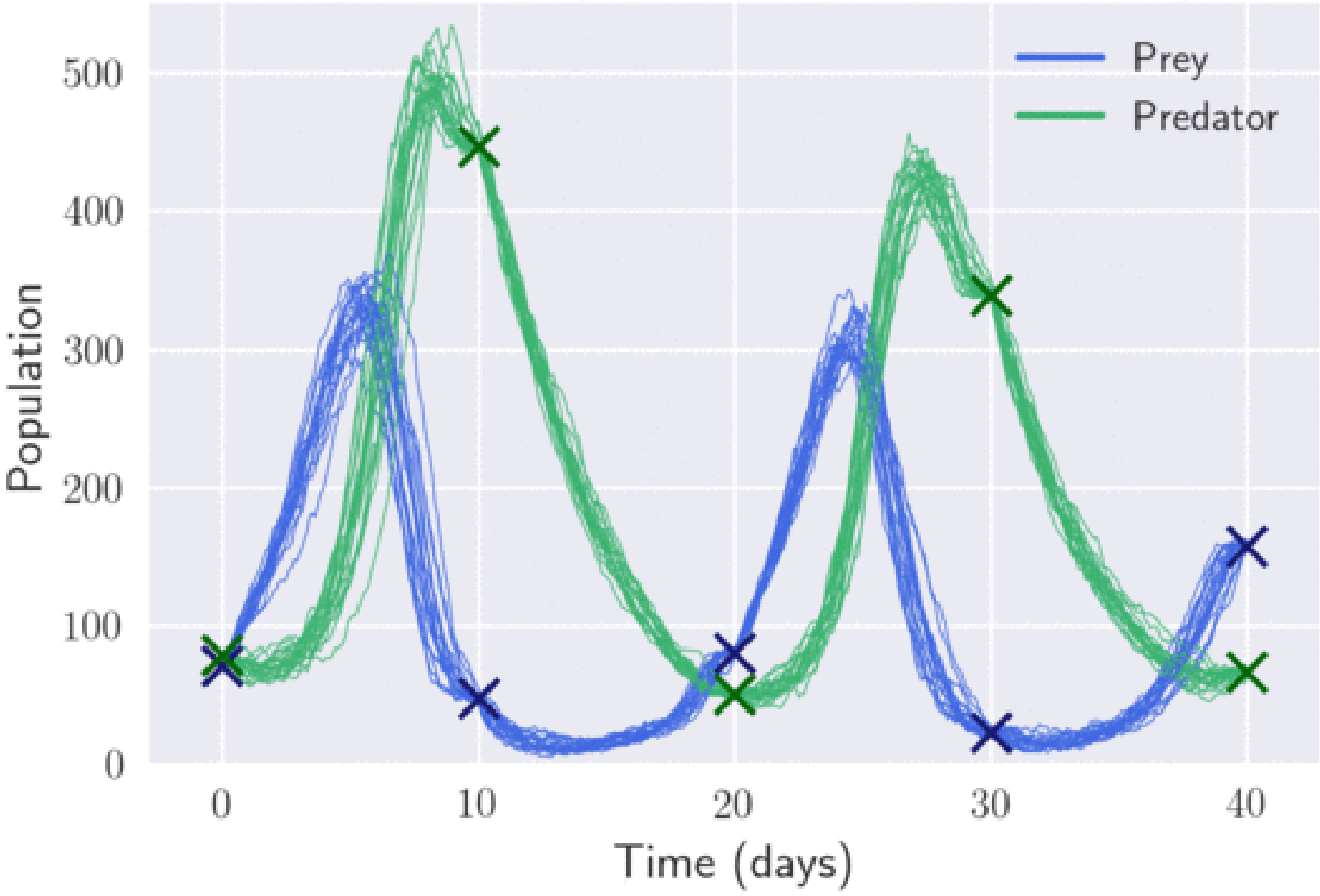
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

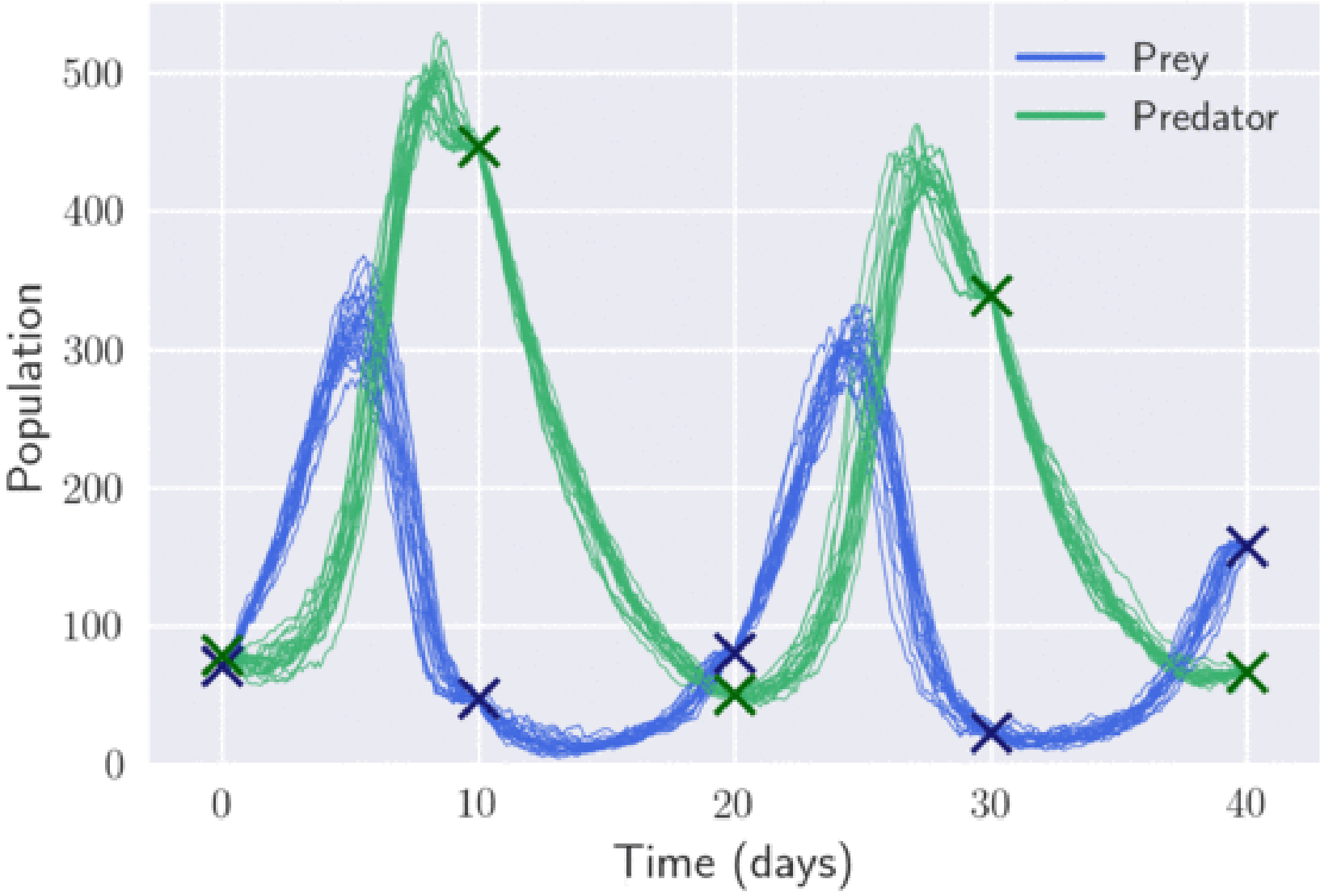
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

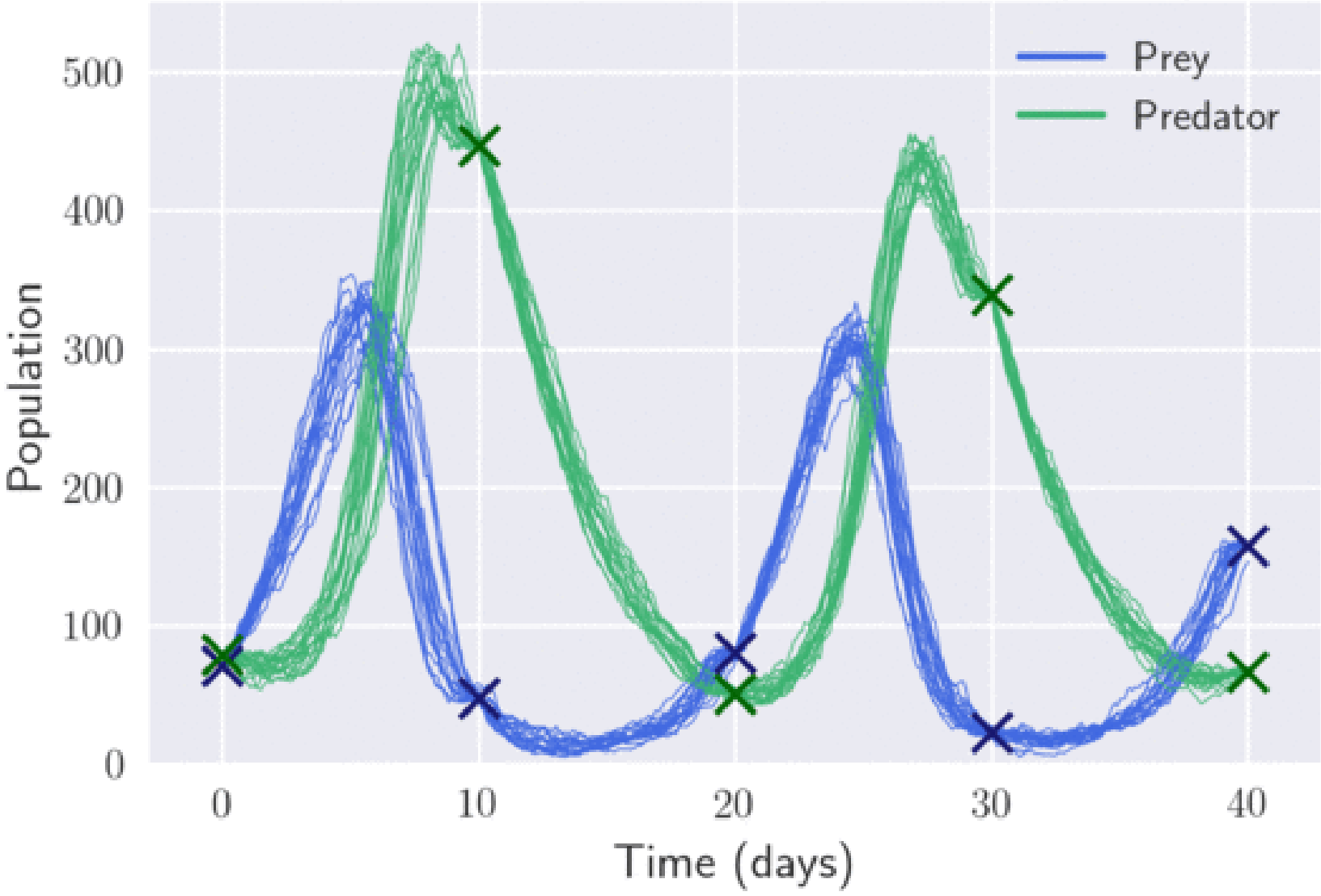
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

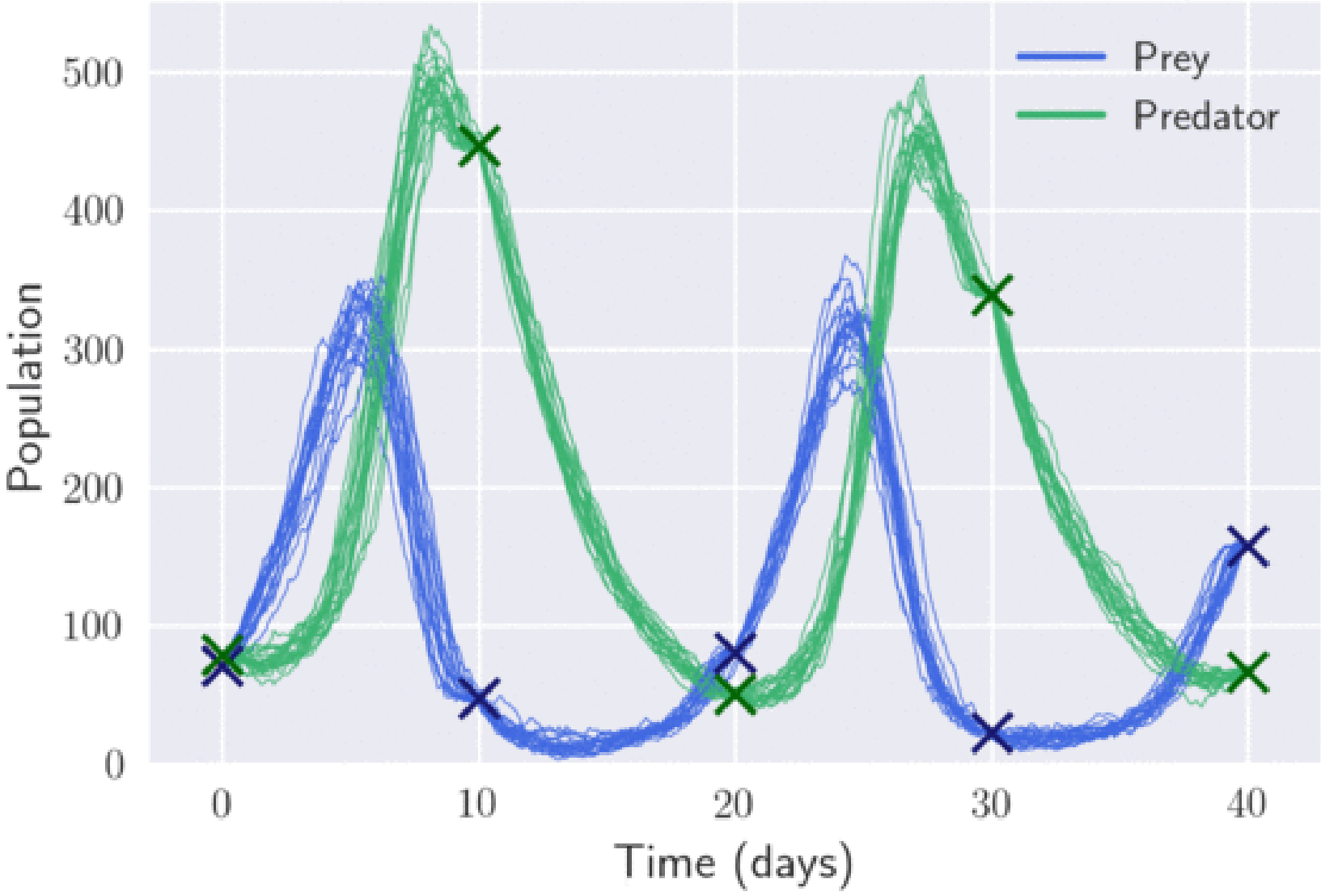
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

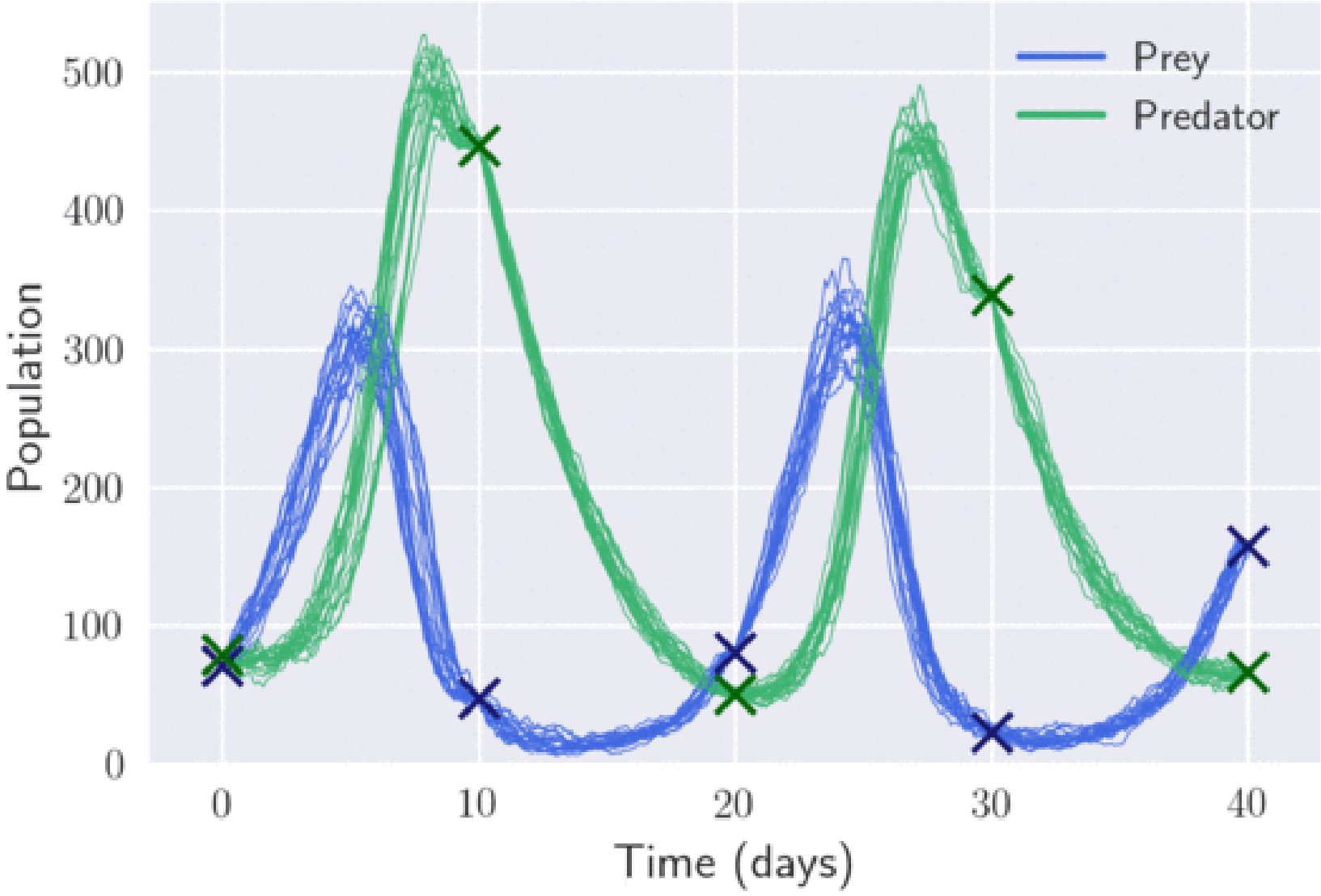
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

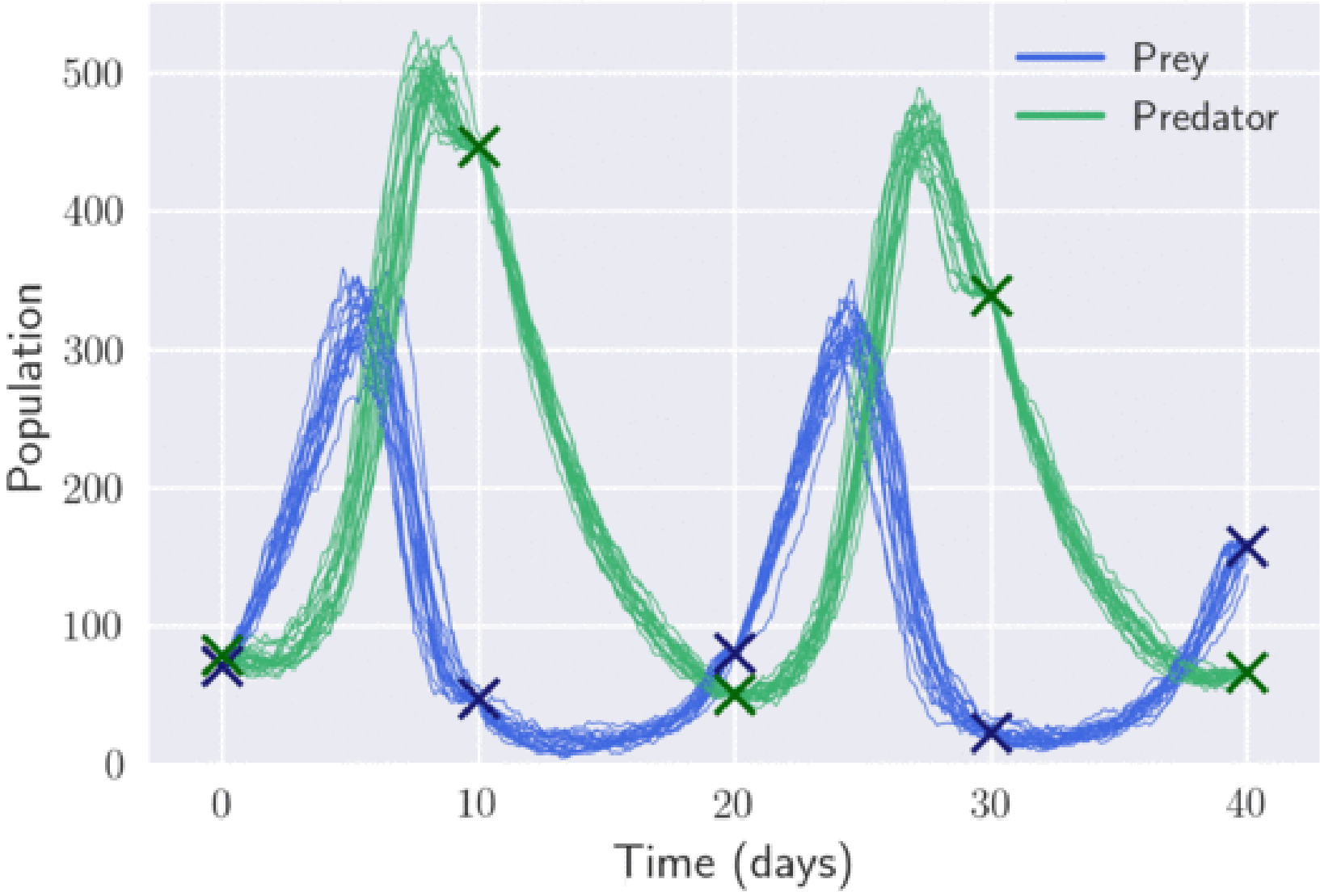
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

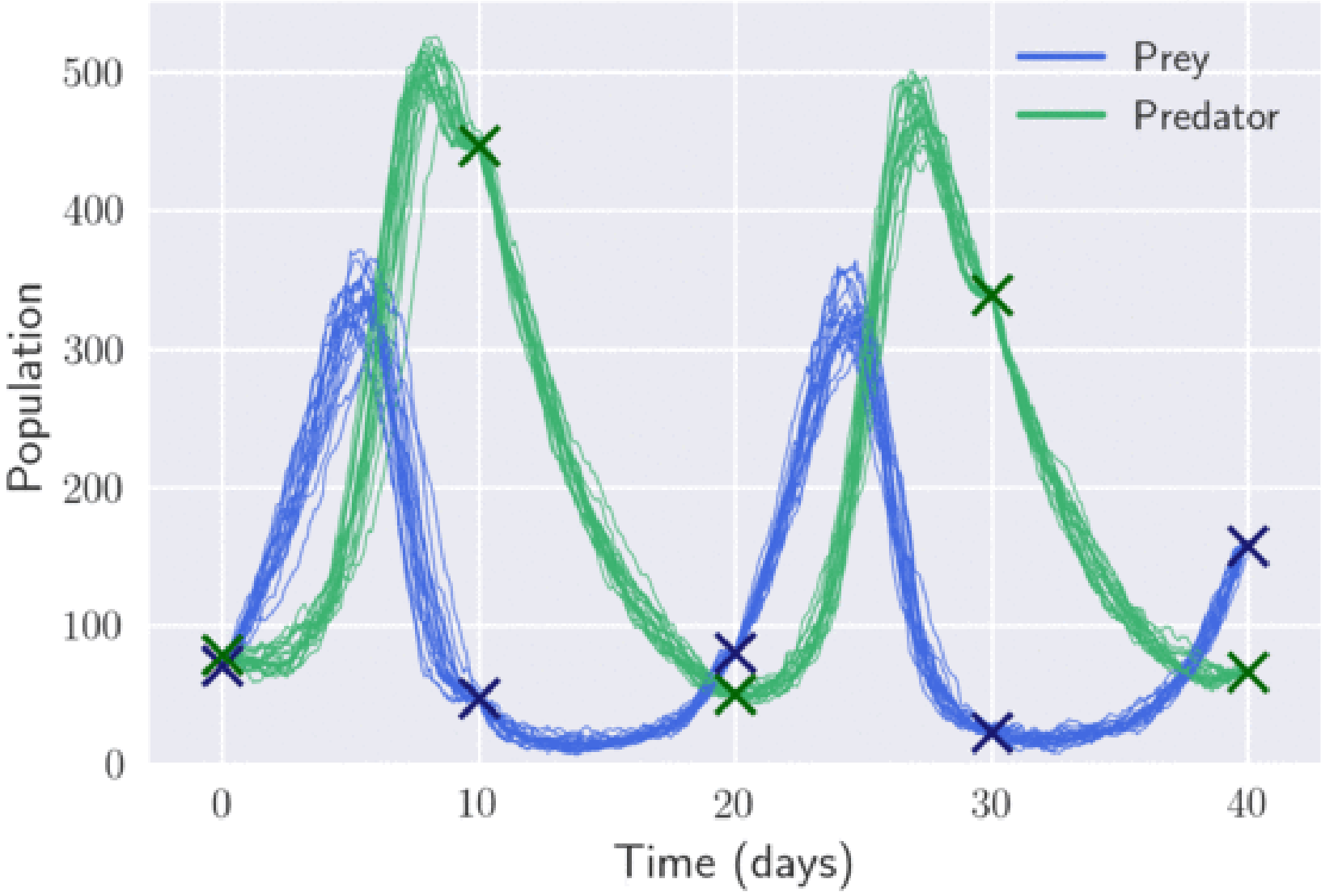
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

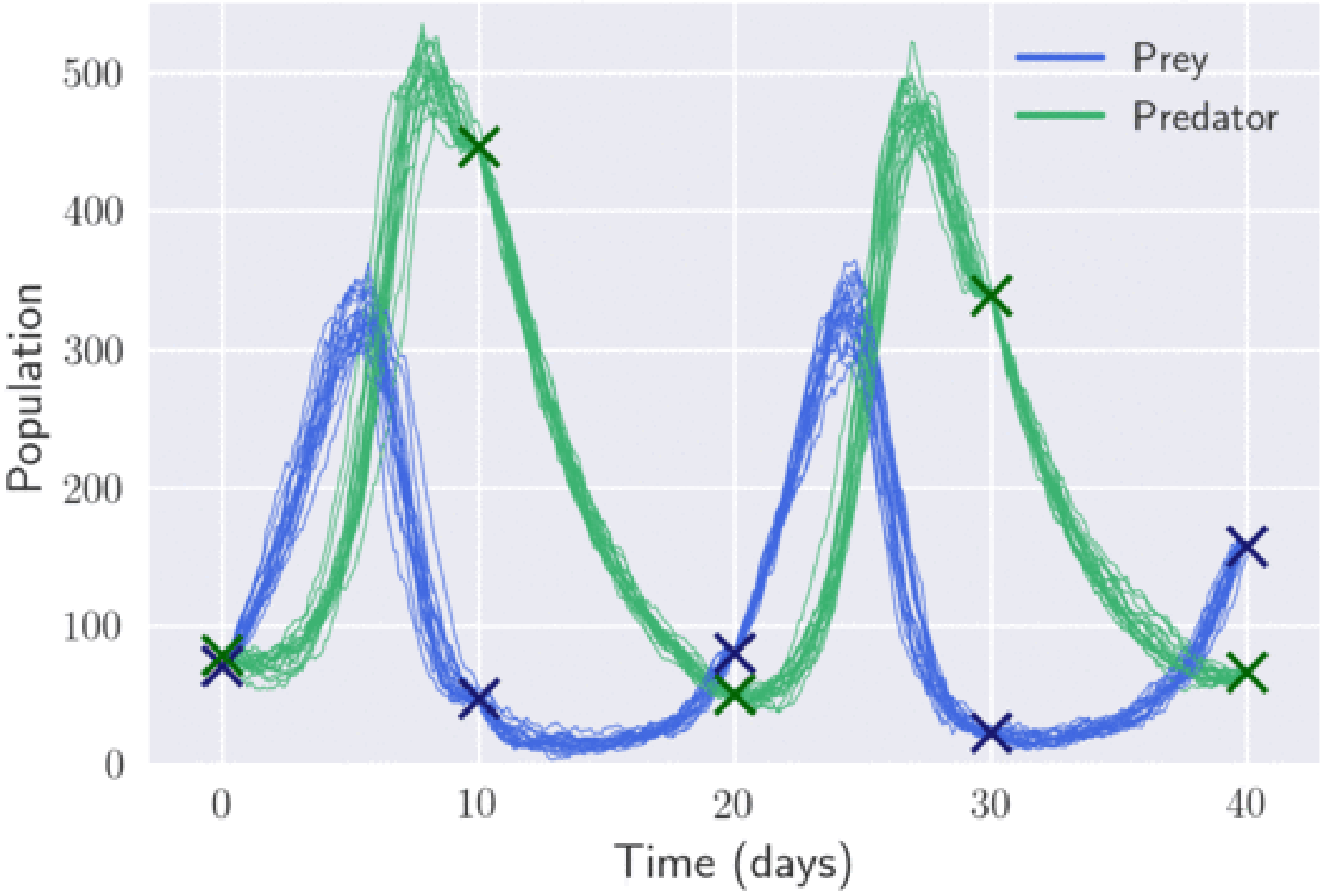
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

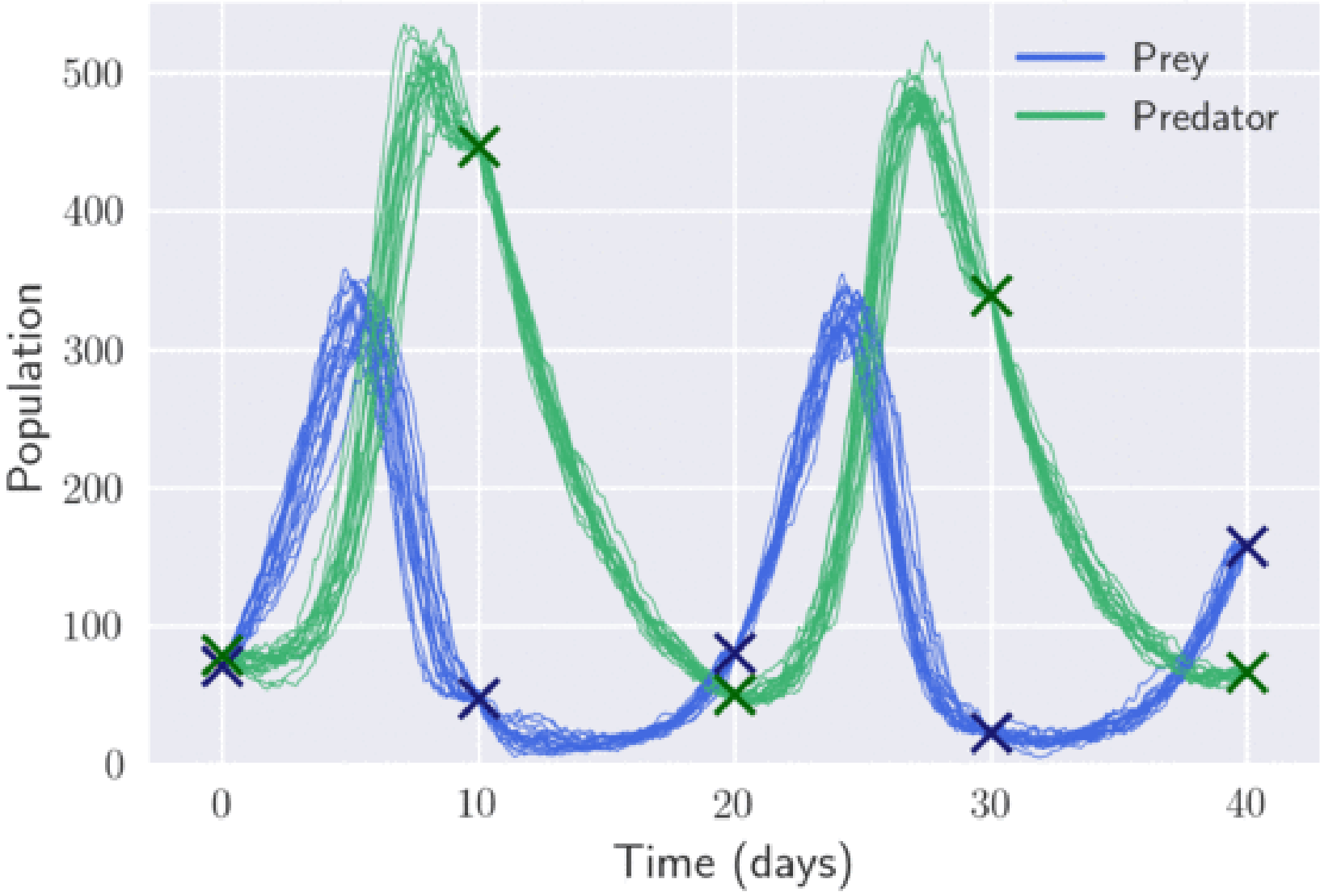
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

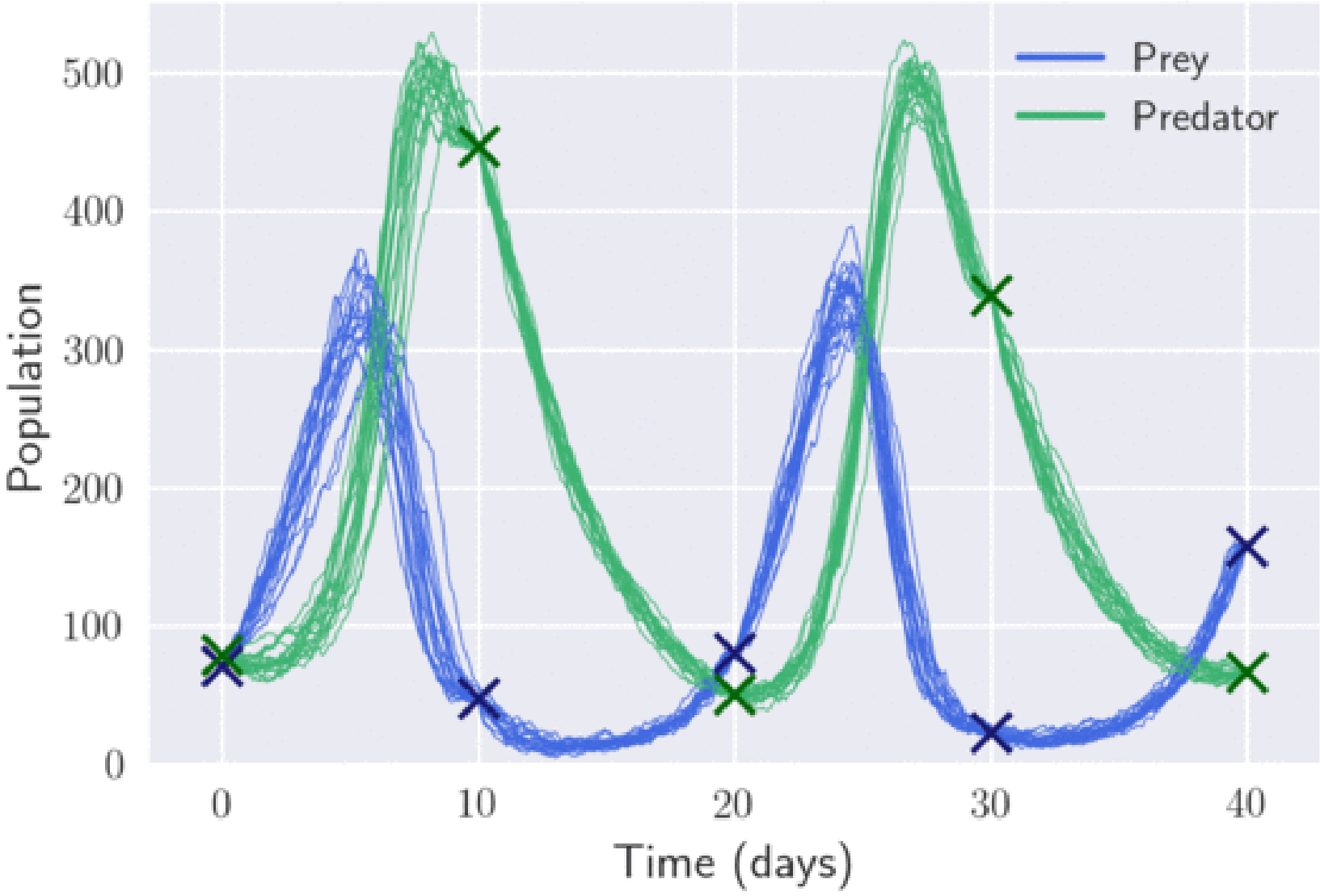
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

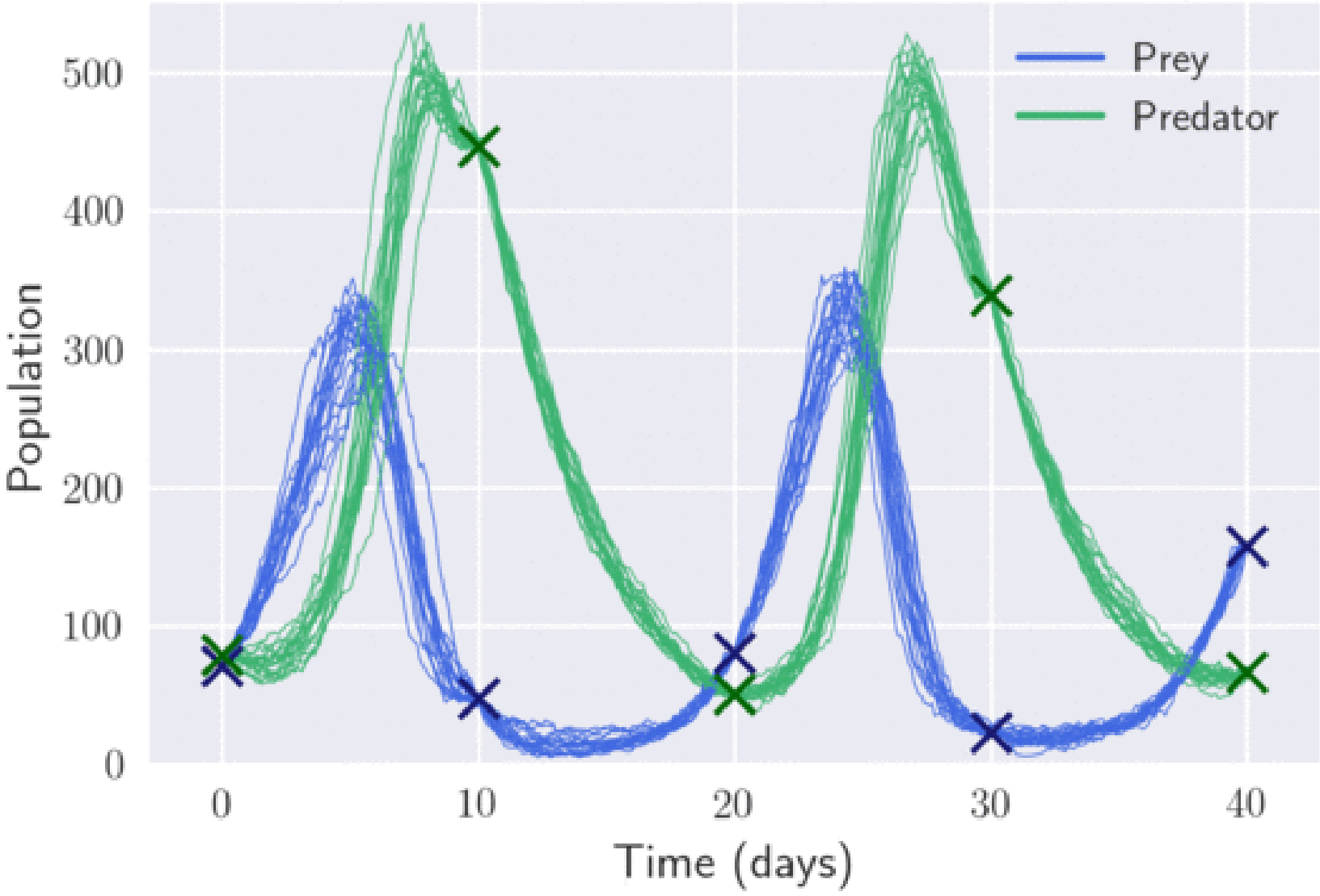
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

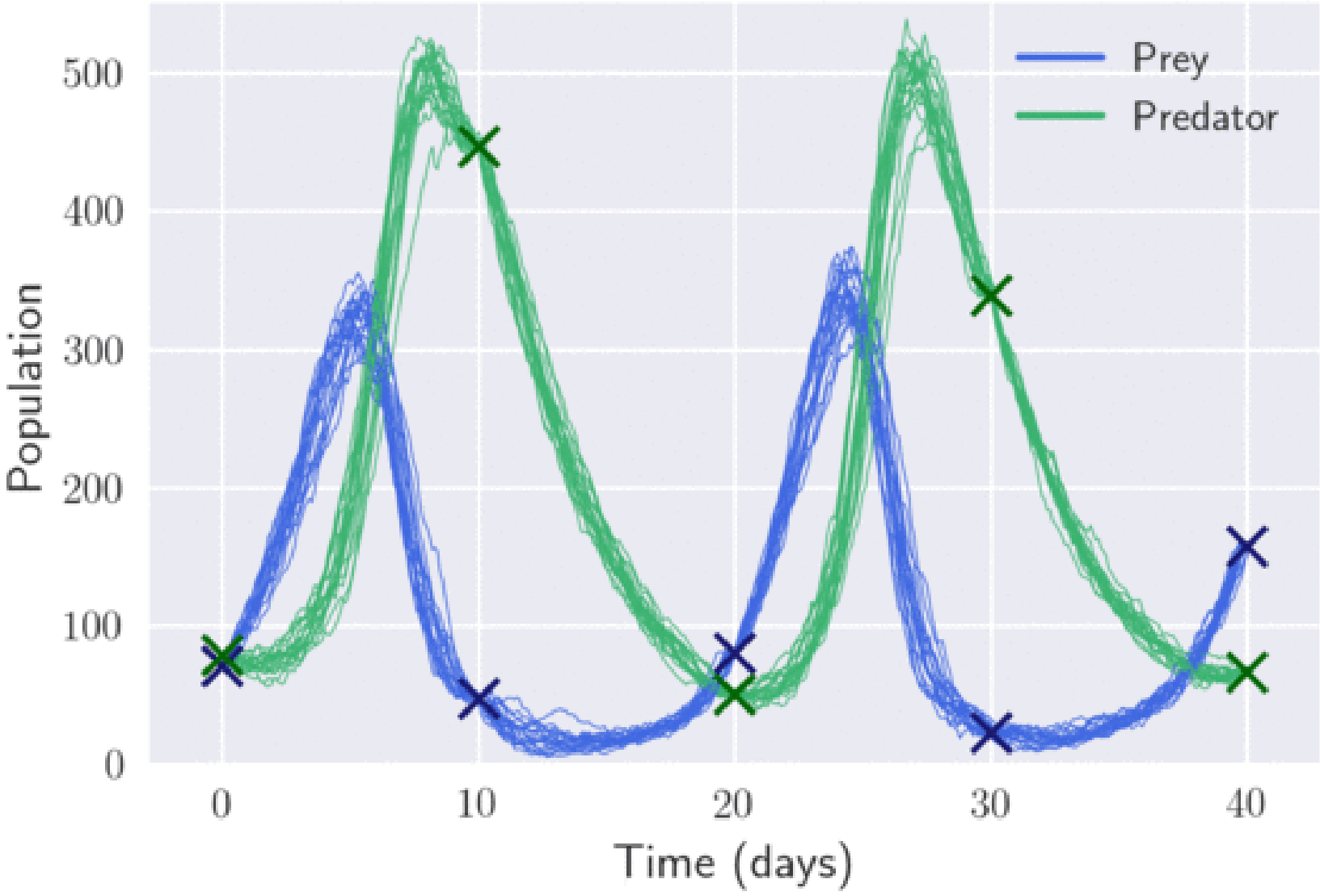
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

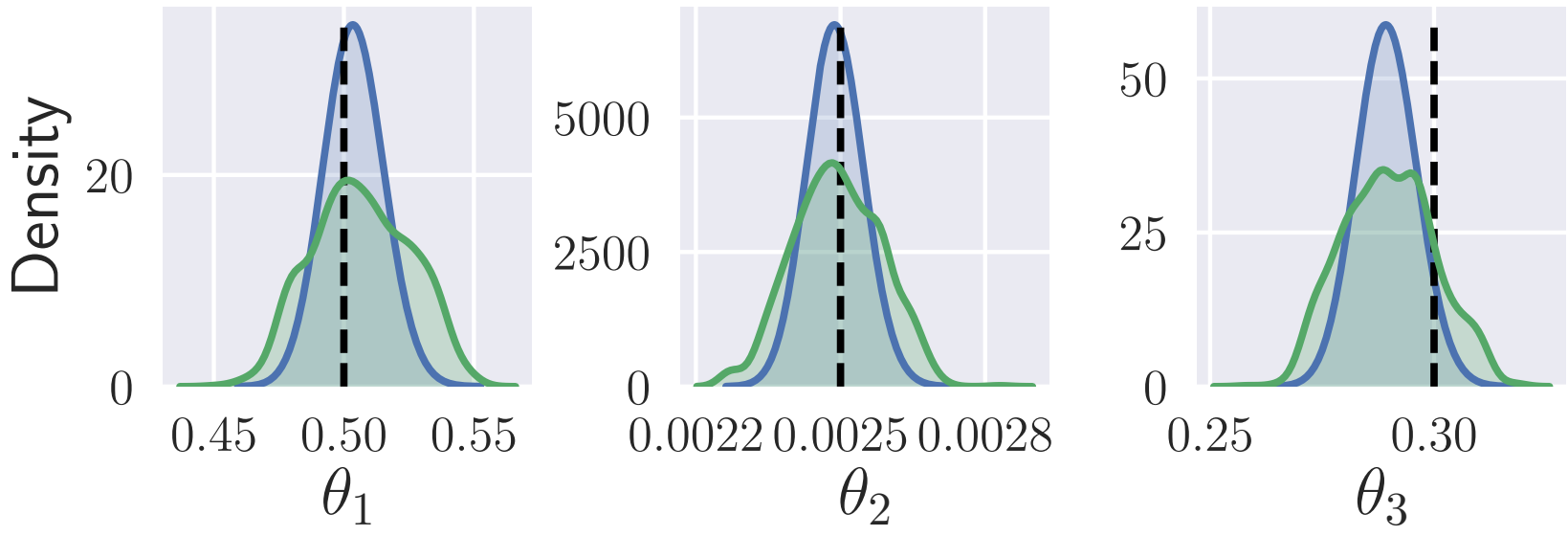
Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

Results


Prey
Predator
Population
Time (days)
0
100
200
300
400
500
0
10
20
30
40

Parameter inference results

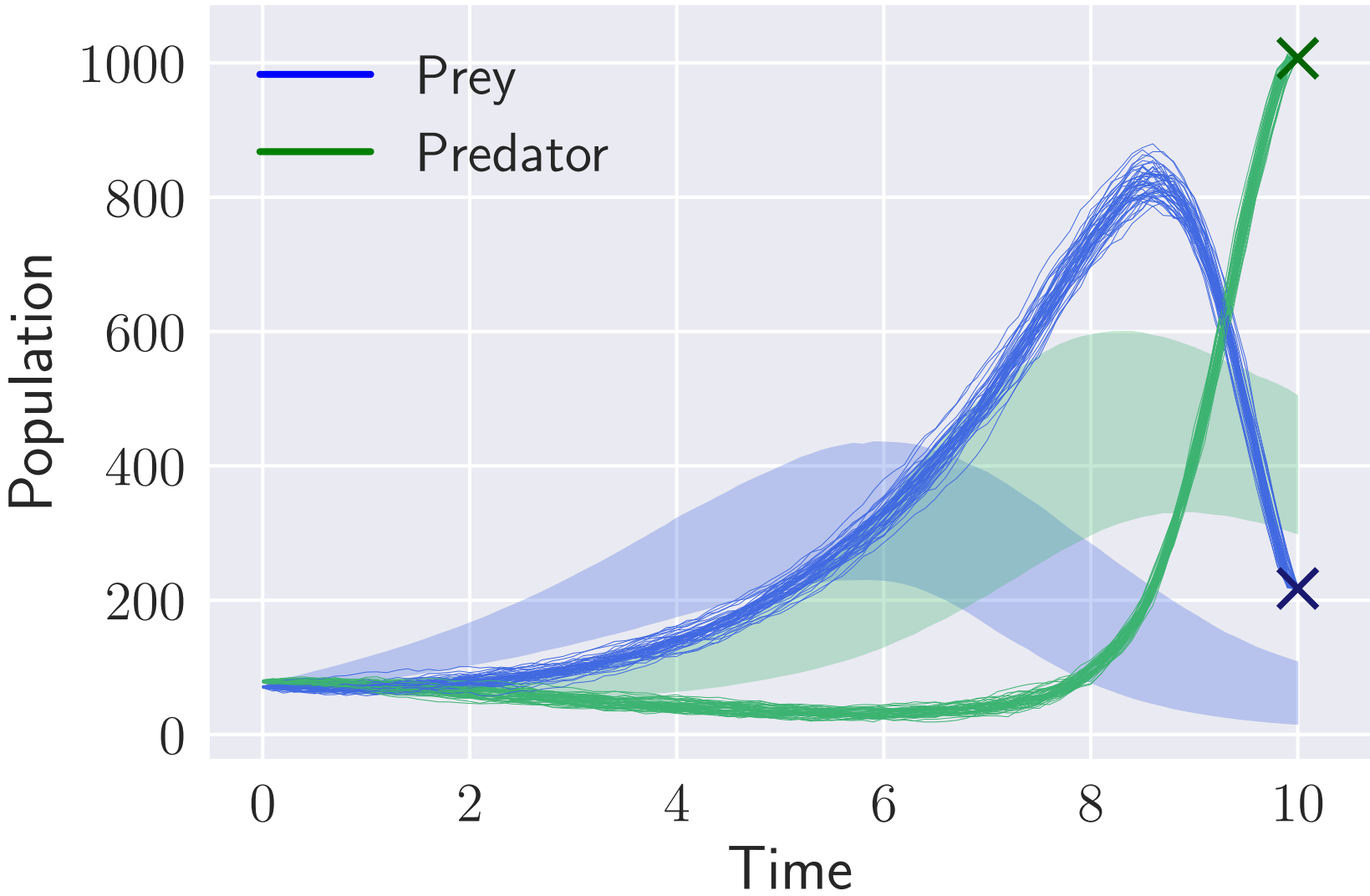


- Black: true parameter values
- Blue: variational output
- Green: importance sampling (shows over-concentration)

Computing time: $\approx$ 2 hours on a desktop PC

Results

Can also do inference under misspecifed model:



Conclusion

Summary

- Variational approach to approx Bayesian inference for SDEs
- Little tuning required
- Results in a few hours on a desktop PC
- We observe good estimation of posterior mode

More in paper

- Real data epidemic example
- Diffusions with unobserved components

Current/future work

- Normalising flows instead of RNNs
- Big data - long or wide
- Other models: state space models, HMMs, MJPs...
  (discrete variables challenging!)
- Model comparison/improvement
- Real applications - **get in touch with any suggestions!**

Acknowledgements and reference

- Joint work with Tom Ryder, Steve McGough, Andy Golightly



- Supported by EPSRC cloud computing for big data CDT
- and NVIDIA academic GPU grant

- http://proceedings.mlr.press/v80/ryder18a.html
- https://github.com/Tom-Ryder/VIforSDEs
- Presented at ICML 2018