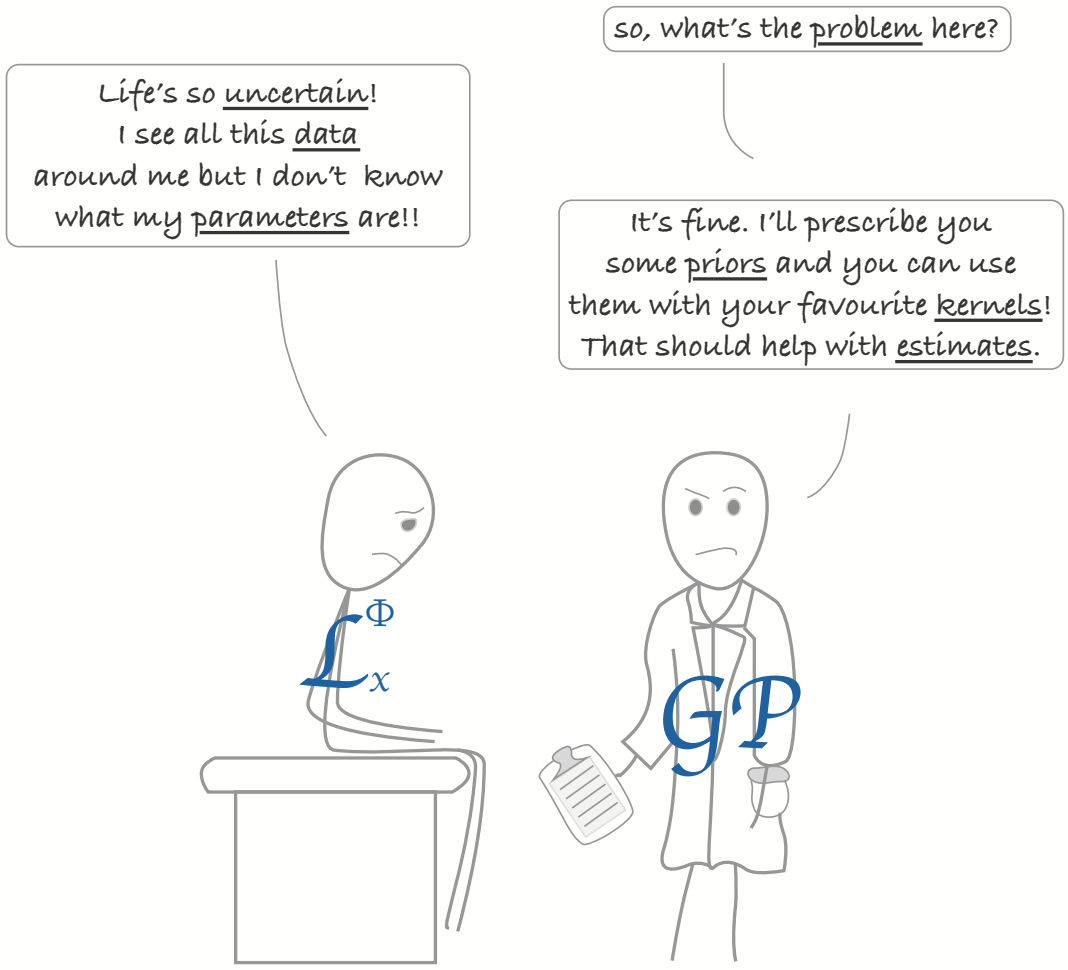
DI-LAB Final Presentation

Parameter Estimation with Gaussian Processes

A. Grundner, K. Wang, K. Harsha

Scientific Lead: Prof. Dr. Eric Sonnendrücker, Dr. Ahmed Ratnani

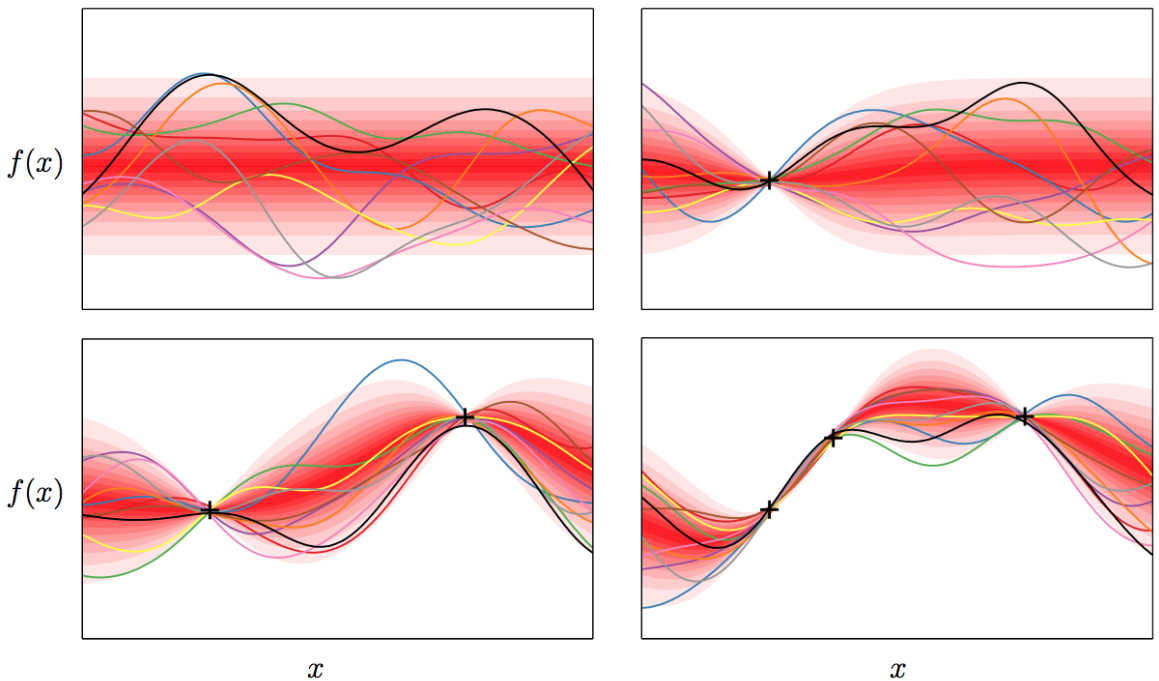
How it started!



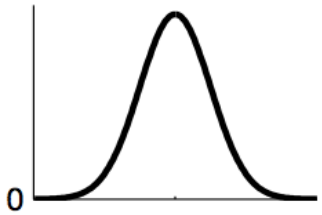
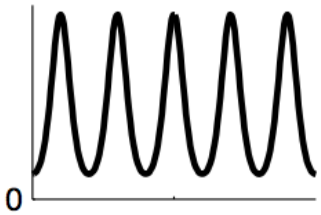
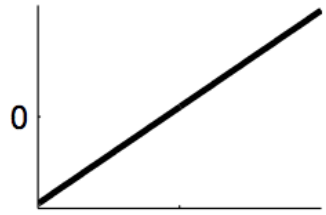
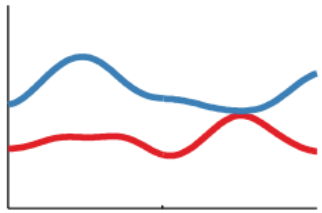
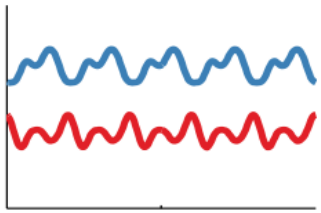
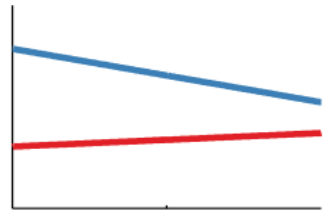
Contents

Gaussian Processes

- A **stochastic process** with a distribution over functions
- Specified by a **mean function**, $m(x)$, and a **covariance function**, or kernel, $k(x, x')$
- Application in machine learning: regression, classification...

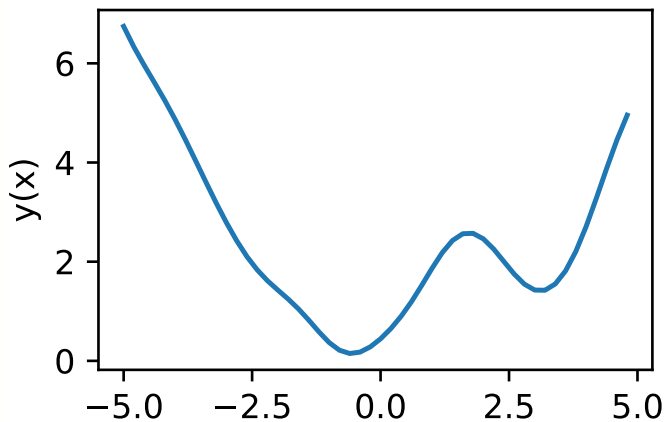


Different Kernels

Kernel name:	Squared-exp (SE)	Periodic (Per)	Linear (Lin)
$k(x, x') =$	$\sigma_f^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$	$\sigma_f^2 \exp\left(-\frac{2}{\ell^2}\sin^2\left(\pi\frac{x-x'}{p}\right)\right)$	$\sigma_f^2(x-c)(x'-c)$
Plot of $k(x, x')$:			
	$x - x'$	$x - x'$	x (with $x' = 1$)
Functions $f(x)$ sampled from GP prior:			
Type of structure:	local variation	repeating structure	linear functions

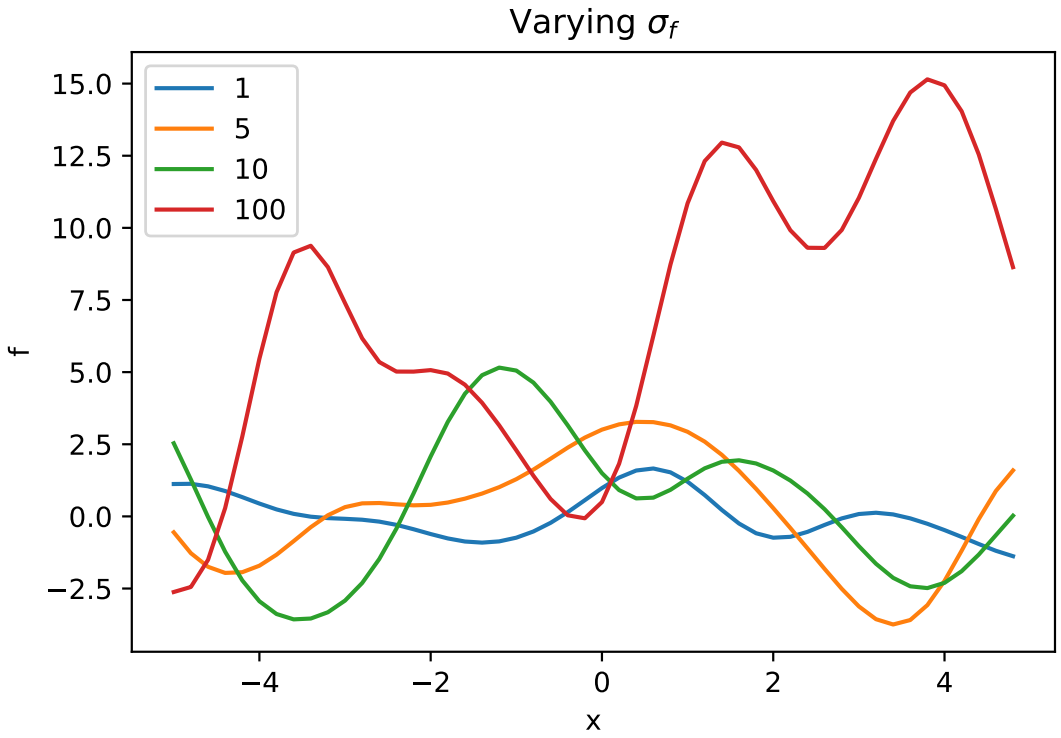
Realization of a GP

$$f \sim \mathcal{GP}(m, k)$$
$$m(x) = \frac{x^2}{4}$$
$$k(x, x') = exp(-\frac{1}{2}(x - x')^2)$$
$$y = f + \epsilon$$
$$\epsilon \sim \mathcal{N}(0, \sigma^2)$$



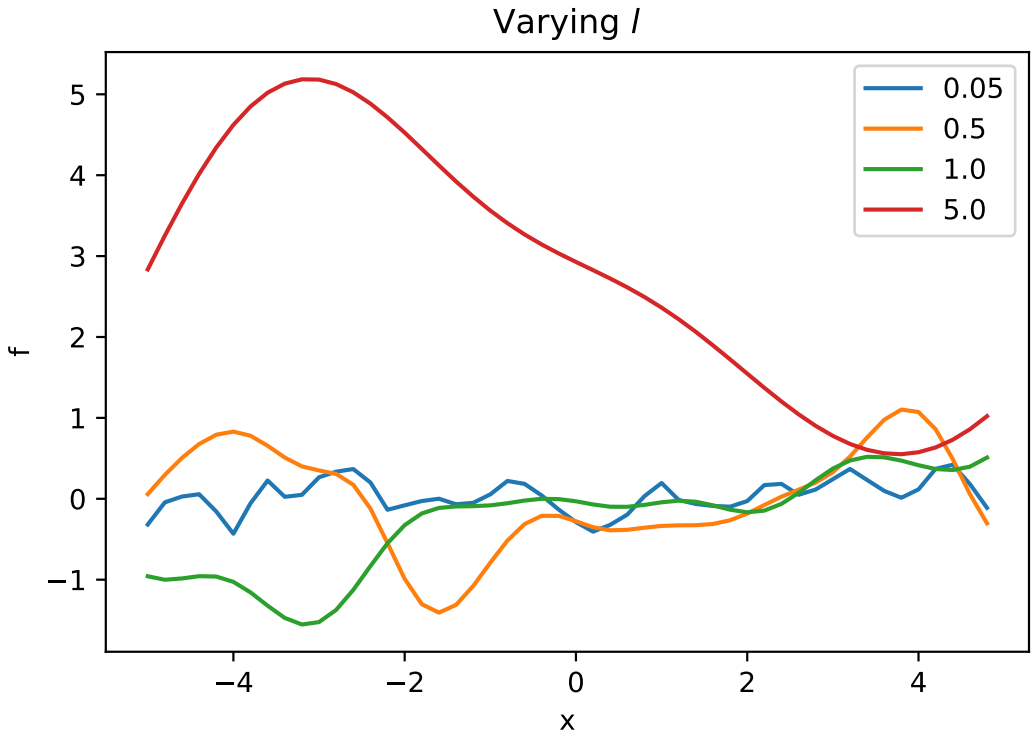
Varying signal variance σ_f for SE kernel

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$



Varying length scale / for SE kernel

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$



Contents

Framework for parameter estimation

$$\mathcal{L}_{\bar{x}}^{\phi} u = f$$
$$u \sim \mathcal{GP}(0, k_{uu}(\bar{x}_i, \bar{x}_j, \theta))$$
$$f \sim \mathcal{GP}(0, k_{ff}(\bar{x}_i, \bar{x}_j, \theta, \phi))$$
$$k_{ff}(\bar{x}_i, \bar{x}_j; \theta, \phi) = \mathcal{L}_{\bar{x}_i}^{\phi} \mathcal{L}_{\bar{x}_j}^{\phi} k_{uu}(\bar{x}_i, \bar{x}_j; \theta)$$
$$k_{fu}(\bar{x}_i, \bar{x}_j; \theta, \phi) = \mathcal{L}_{\bar{x}_i}^{\phi} k_{uu}(\bar{x}_i, \bar{x}_j; \theta)$$

Dataset: $\bar{x}, \bar{y}_u, \bar{y}_f$

Framework for parameter estimation

$$y = \begin{bmatrix} \bar{y}_u \\ \bar{y}_f \end{bmatrix}$$

$$K = \begin{bmatrix} k_{uu}(\bar{x}_u, \bar{x}_u; \theta) + \sigma_u^2 I & k_{uf}(\bar{x}_u, \bar{x}_f; \theta, \phi) \\ k_{fu}(\bar{x}_f, \bar{x}_u; \theta, \phi) & k_{ff}(\bar{x}_f, \bar{x}_f; \theta, \phi) + \sigma_f^2 I \end{bmatrix}$$

$$\mathcal{NLML} = \frac{1}{2}\left[log|K| + y^T K^{-1} y + Nlog(2\pi)\right]$$

Take maximum likelihood estimates: θ_{est}, ϕ_{est}

1D Linear operator with more than one parameter

$\mathcal{L}_x^{\phi} u(x) = f(x)$

$\mathrm{L}_x^{\phi} := \phi_1 \cdot + \phi_2 \frac{d}{dx} \cdot$

u (x) = $\sin(x)$

f (x) = $\phi_1 \sin(x) + \phi_2 \cos(x)$

$k_{ff}\left(x_i, x_j; \theta, \phi_1, \phi_2\right)$
$= \mathcal{L}_{x_i}^{\phi} \mathcal{L}_{x_j}^{\phi} k_{uu}\left(x_i, x_j; \theta\right)$
$= \mathcal{L}_{x_i}^{\phi}\left(\phi_1 k_{uu} + \phi_2 \frac{\partial}{\partial x_j} k_{uu}\right)$
$= \phi_1^2 k_{uu} + \phi_1 \phi_2 \frac{\partial}{\partial x_j} k_{uu}$
$+ \phi_1 \phi_2 \frac{\partial}{\partial x_i} k_{uu} + \phi_2^2 \frac{\partial}{\partial x_i} \frac{\partial}{\partial x_i} k_{uu}$

$k_{fu}(x_i, x_j; \theta, \phi_1, \phi_2)$
$= \mathrm{L}_{x_i}^{\phi} k_{uu}(x_i, x_j; \theta)$
$= \phi_1 k_{uu} + \phi_2 \frac{\partial}{\partial x_i} k_{uu}$

$\mathcal{L}_x^{\phi} u(x) = f(x)$

$\mathrm{L}_x^{\phi} := \phi_1 \cdot + \phi_2 \frac{d}{dx} \cdot + \phi_3 \frac{d^2}{dx^2}$

u (x) = $\sin(x)$

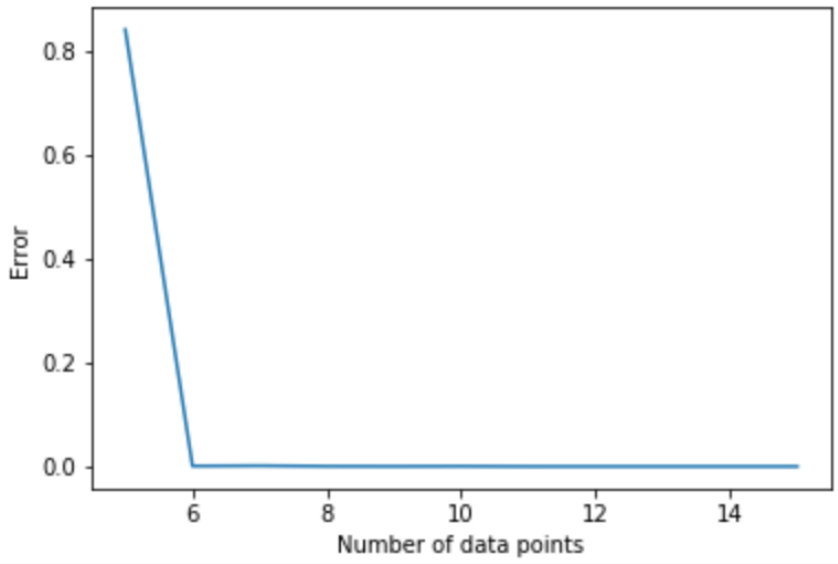
f (x) = $\phi_1 \sin(x) + \phi_2 \cos(x) - \phi_3 \sin(x)$

$k_{ff}\left(x_i, x_j; \theta, \phi_1, \phi_2, \phi_3\right)$
$= \mathcal{L}_{x_i}^{\phi} \mathcal{L}_{x_j}^{\phi} k_{uu}\left(x_i, x_j; \theta\right)$
$= \mathcal{L}_{x_i}^{\phi}\left(\phi_1 k_{uu} + \phi_2 \frac{\partial}{\partial x_j} k_{uu} + \phi_3 \frac{\partial^2}{\partial x_j^2} k_{uu}\right)$
$= \left(\phi_1 k_{uu} + \phi_2 \frac{\partial}{\partial x_i} k_{uu} + \phi_3 \frac{\partial^2}{\partial x_i^2} k_{uu}\right)$
$\left(\phi_1 k_{uu} + \phi_2 \frac{\partial}{\partial x_j} k_{uu} + \phi_3 \frac{\partial^2}{\partial x_j^2} k_{uu}\right)$

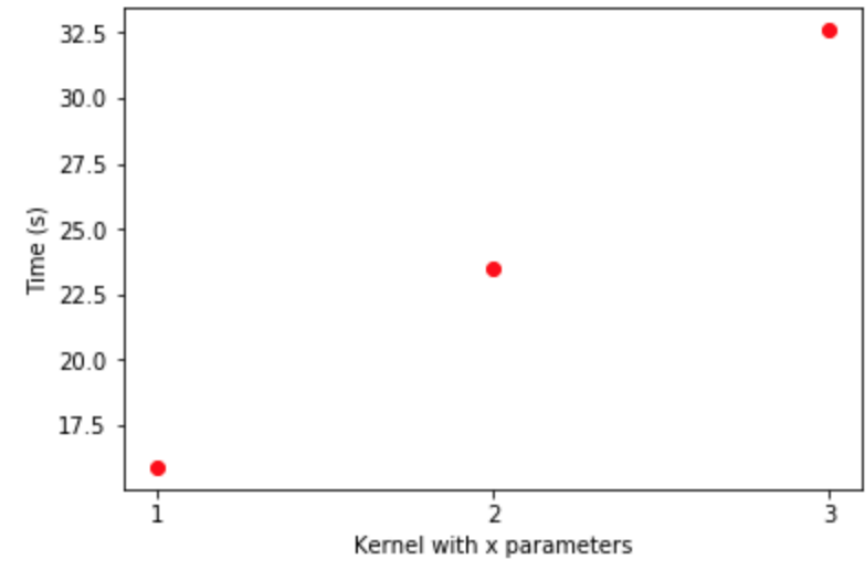
$k_{fu}\left(x_i, x_j; \theta, \phi_1, \phi_2, \phi_3\right)$
$= \mathcal{L}_{x_i}^{\phi} k_{uu}\left(x_i, x_j; \theta\right)$
$= \phi_1 k_{uu} + \phi_2 \frac{\partial}{\partial x_i} k_{uu} + \phi_3 \frac{\partial^2}{\partial x_i^2} k_{uu}$

1D Linear operator with more than one parameter

$\phi_1 = 2, \phi_2 = 5$



$\phi_2 = 5$



Contents

Heat equation

$$\frac{\partial u}{\partial t} - \phi \nabla^2 u = f$$

In one spatial dimension,

$$\mathcal{L}^{\phi}_{\bar{x}} u(\bar{x}) = \frac{\partial}{\partial t} u(\bar{x}) - \phi \frac{\partial^2}{\partial x^2} u(\bar{x}) = f(\bar{x}),$$

where $\bar{x} = (t, x) \in \mathbb{R}^2$

Backward Euler scheme

For the homogeneous case:

$$u_t - \alpha u_{xx} = 0$$

$$u(x,t) = e^{-t} sin(\frac{x}{\sqrt{\alpha}})$$

$$u_0(x) := u(x,0) = sin(\frac{x}{\sqrt{\alpha}})$$

Discretization in the time domain:

$$\frac{u_n - u_{n-1}}{\tau} - \alpha \frac{d^2}{dx^2} u_n = 0$$

$$u_n - \tau\alpha \frac{d^2}{dx^2} u_n = u_{n-1}$$

Backward Euler (contd...)

Gaussian prior:

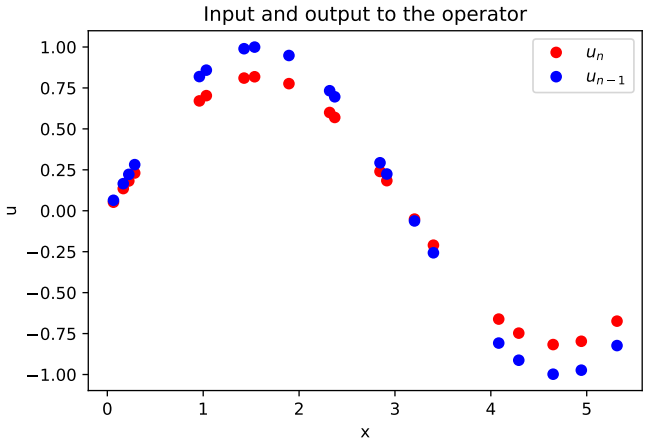
$$u_n \sim \mathcal{GP}(0, k_{uu}(x_i, x_j, \theta))$$

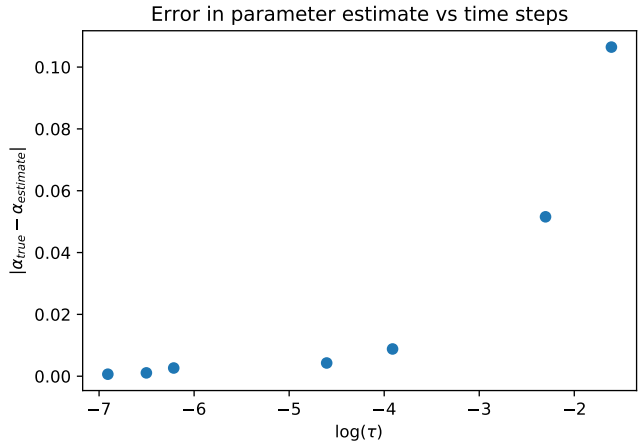
Linear operator:

$$\mathcal{L}_x^\alpha = \cdot - \tau\alpha\frac{d^2}{dx^2}\cdot$$
$$\mathcal{L}_x^\alpha u = f; \quad u := u_n, f := u_{n-1}$$

Backward Euler scheme: Results

Kernel: $k_{uu}(x_i, x_j; \theta) = e^{(\theta(x_i - x_j)^2)}$



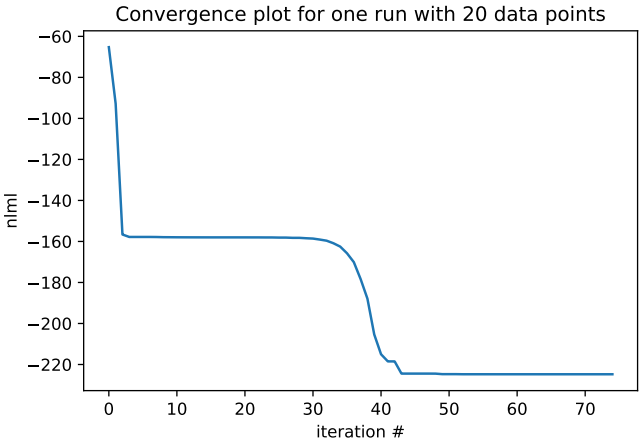



```
%%timeit
minimize(nlml,
         np.random.rand(2),
         args=(x, yu, yf, 0.2, 1e-6),
         method="Nelder-Mead")
```

```
3.43 s ± 275 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
nlml_wp = lambda params: nlml(params, x, yu, yf, 0.2, 1e-6)
param_bounds = np.array([[-3,0], [-5,1]])
%timeit minimize(nlml_wp,
                 np.random.rand(2),
                 method="TNC",
                 bounds = param_bounds)
```

```
The slowest run took 12.22 times longer than the fastest. This could
d.
35.3 s ± 44.1 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```



Backward Euler for the nonhomogeneous case

$$
\begin{aligned}
u_t - \alpha u_{xx} &= f \\
u(x,t) &= e^{-t} sin(2\pi x), \quad u_0(x) := u(x,0) = sin(2\pi x) \\
f(x,t) &= (-1 + 4\alpha\pi^2) e^{-t} sin(2\pi x) \\
x, t &\in [0,1]
\end{aligned}
$$

with the Backward Euler scheme:

$$
\begin{aligned}
\frac{u_n - u_{n-1}}{\tau} - \alpha \frac{d^2}{dx^2} u_n &= f_n \\
u_n - \tau\alpha \frac{d^2}{dx^2} u_n &= u_{n-1} + \tau f_n
\end{aligned}
$$

Backward Euler for the nonhomogeneous case (contd...)

Gaussian prior:

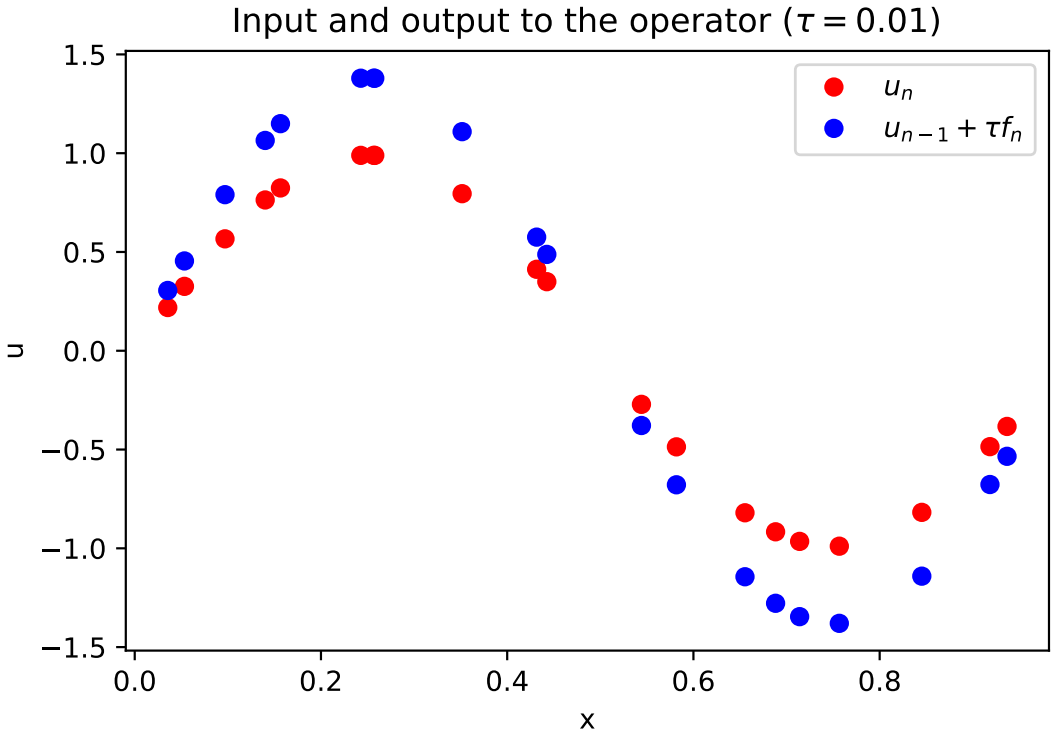
$$u_n \sim \mathcal{GP}(0, k_{uu}(x_i, x_j, \theta))$$

Linear operator:

$$\mathcal{L}_x^\alpha = \cdot - \tau\alpha\frac{d^2}{dx^2}\cdot$$
$$\mathcal{L}_x^\alpha u = f; \quad u := u_n, f := u_{n-1} + \tau f_n$$

Backward Euler for the nonhomogeneous case: Data

- 20 data points
- $\tau = 0.01$



Backward Euler for the nonhomogeneous case: Results

Comparison between kernels:

For $k_{uu}(x_i, x_j; \theta) = e^{\theta(x_i - x_j)^2}$

```
%%timeit
minimize(nlml,
         np.random.rand(2),
         args=(x, y_u, y_f, 0.01, 1e-6),
         method="Nelder-Mead")
```

```
3.04 s ± 237 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

For $k_{uu}(x_i, x_j; \theta) = \theta e^{-\frac{1}{2}(x_i - x_j)^2}$

```
%%timeit
minimize(nlml,
         np.random.rand(2),
         args=(x, y_u, y_f, 0.01, 1e-6),
         method="Nelder-Mead")
```

```
7.54 s ± 1.62 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

General case

$$\mathcal{L}_{\bar{x}}^{\phi} u(\bar{x}) = \frac{\partial}{\partial t} u(\bar{x}) - \phi \frac{\partial^2}{\partial x^2} u(\bar{x}) = f(\bar{x}),$$

where $\bar{x} = (t, x) \in \mathbb{R}^2$.
Gaussian prior:

$$u \sim \mathcal{GP}(0, k_{uu}(x_i, x_j, \theta))$$

$$k_{uu}(x_i, x_j, t_i, t_j; \theta) = e^{\left[-\theta_1 (x_i - x_j)^2 - \theta_2 (t_i - t_j)^2\right]}$$

General case: Benchmark

- 10 data points
- Noise variance: 10^{-7}

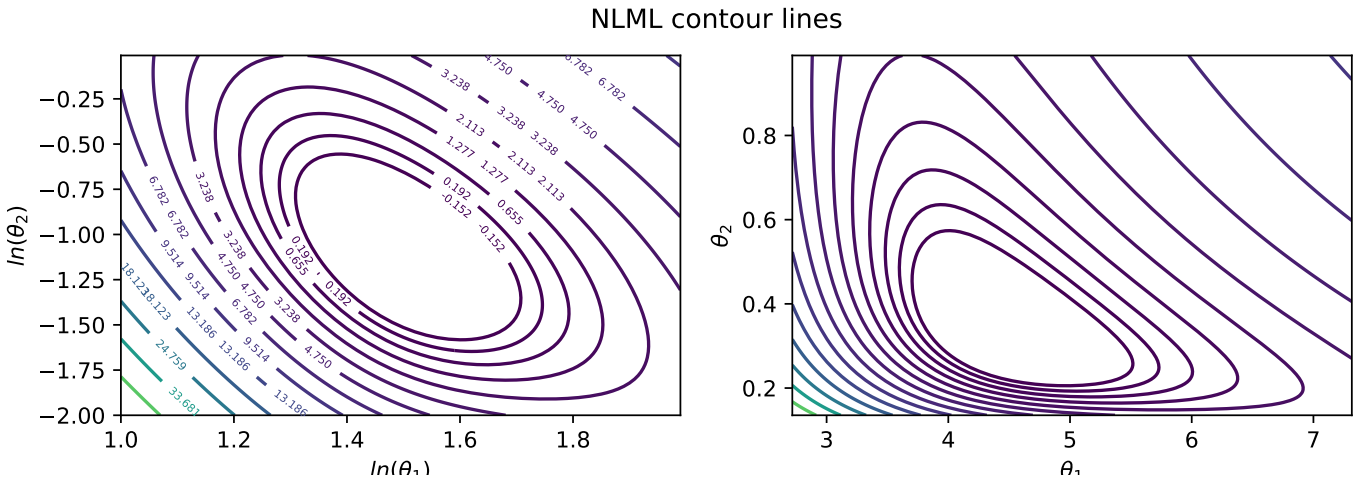
```
%%timeit
nlml_wp = lambda params: nlml(params, t, x, y_u, y_f, 1e-7)
minimize(
    nlml_wp,
    np.random.rand(3),
    method="Nelder-Mead",
    options={'maxiter' : 5000, 'fatol' : 0.001})
```

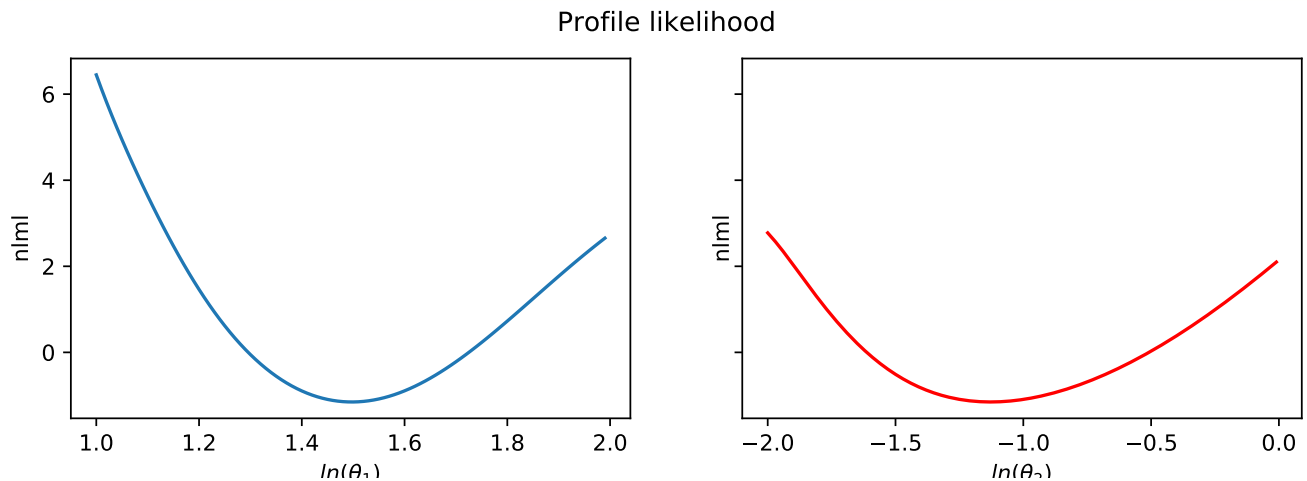
```
2.13 s ± 267 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
np.exp(m.x)
```

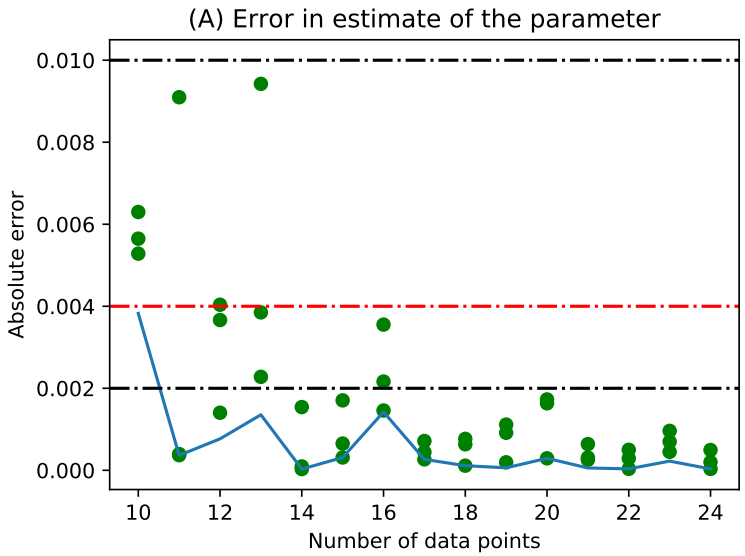
```
array([4.0413258 , 0.28249211, 0.99659146])
```

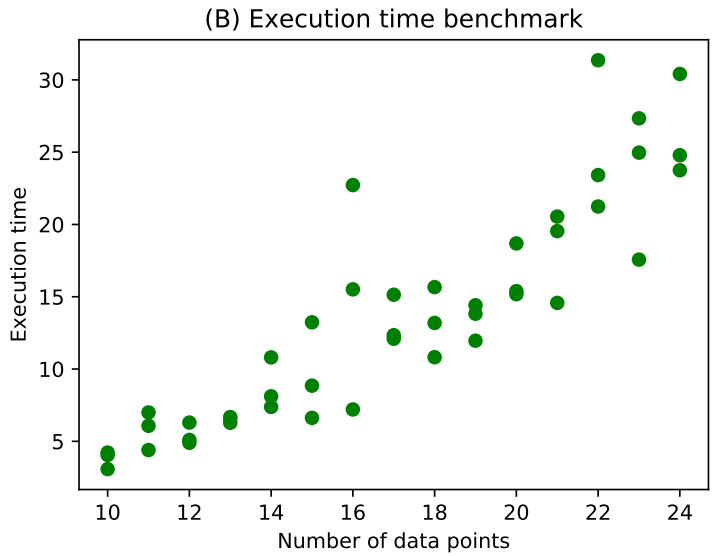
General case: Results





General case: Simulation results





General case: Comparison with the full kernel

Full kernel: $\theta exp(-\frac{1}{2l_1}(x_i - x_j)^2 - \frac{1}{2l_2}(t_i - t_j)^2)$

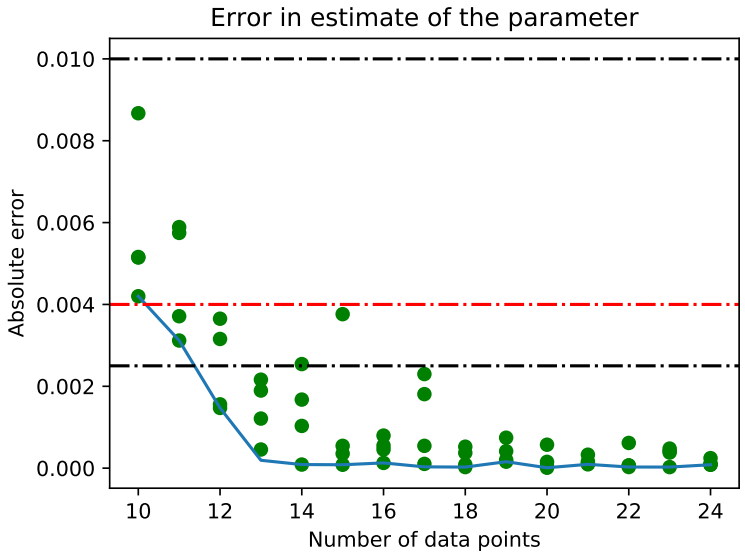
```
%%timeit
nlml_wp = lambda params: nlml(params, t, x, y_u, y_f, 1e-7)
minimize(
    nlml_wp,
    np.random.rand(4),
    method="Nelder-Mead",
    options={'maxiter' : 5000, 'fatol' : 0.001})
```

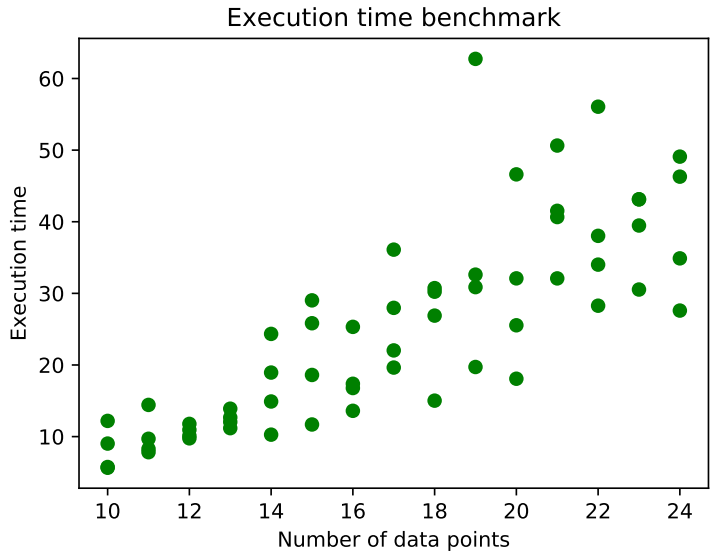
```
6.93 s ± 1.94 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
nlml_wp = lambda params: nlml(params, t, x, y_u, y_f, 1e-7)
m_f = minimize(
    nlml_wp,
    np.random.rand(4),
    method="Nelder-Mead",
    options={'maxiter' : 5000, 'fatol' : 0.001})
np.exp(m_f.x)
```

```
array([1.37659982, 0.132489  , 1.91092044, 0.99181556])
```

General case: Comparison with the full kernel





Contents

The Wave Equation

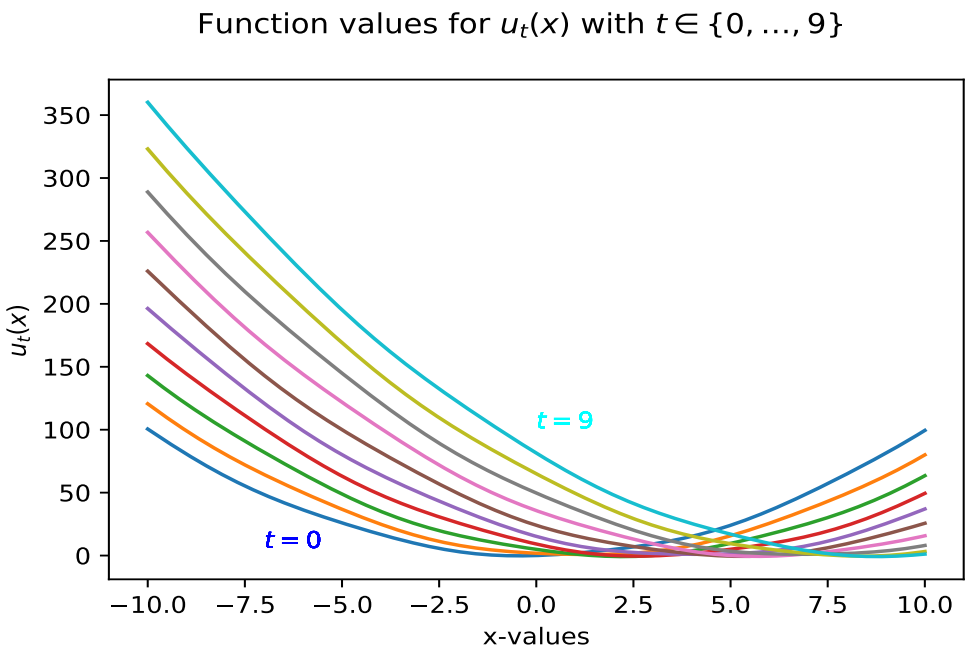
$$\boxed{\frac{\partial^2}{\partial t^2} u = c\nabla^2 u}$$

Can rewrite it (in one spatial dimension) as $\mathcal{L}_x^c u = f$, where $f = 0$ and

$$\mathcal{L}_x^c = \frac{\partial^2}{\partial t^2} \cdot - c\frac{\partial^2}{\partial x^2} \cdot$$

A solution for $c = 1$:

$$u(x, t) = (x - t)^2 + \sin(x + t).$$



Sample 20 random points X in $[0, 1]^2$ along with $u(X)$ and $f(X)$.

Problem at hand: Estimate c from these samples.

Applying our algorithm

Assumption:

$$u \sim \mathcal{GP}(0, k_{uu}(x_i, x_j; \tilde{\theta})_{i,j}),$$

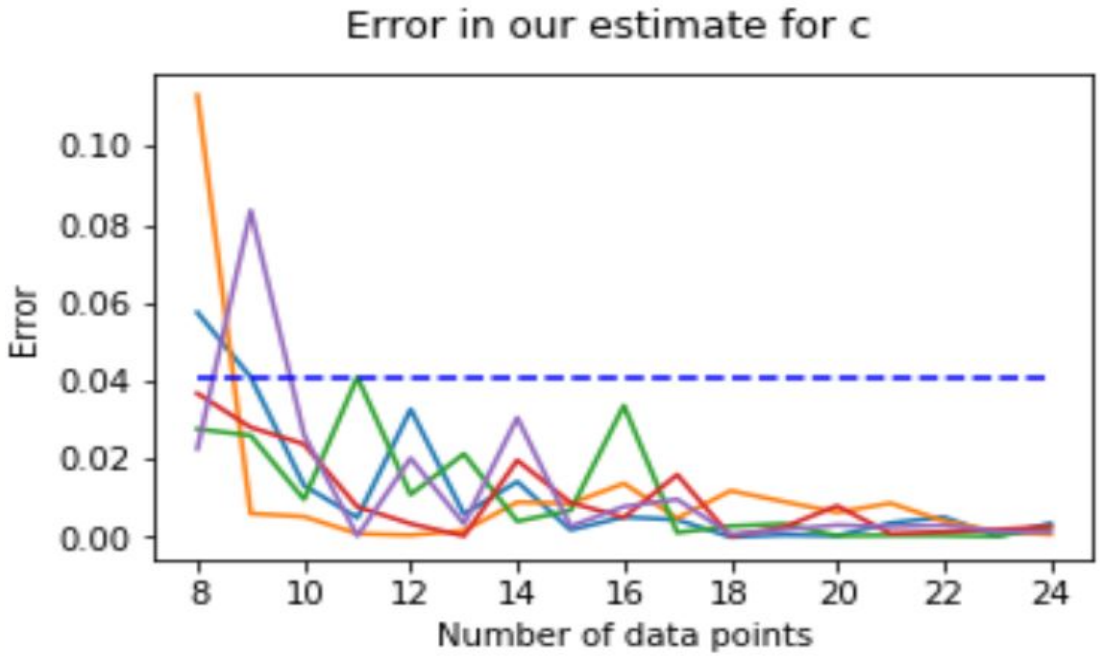
where k_{uu} is an RBF Kernel and $\tilde{\theta} = \{\sigma_u, l_x, l_t\}$.

- f is GP-distributed as a linear transformation of u.
- Minimize the *nlml*, that corresponds to u, f and our data samples.

Our result: $c = 1.0003$

The absolute error in our estimate

We plot the error $|\tilde{c} - 1|$ using five differently colored runs of our algorithm ($\tilde{c}$ is our estimate for c).



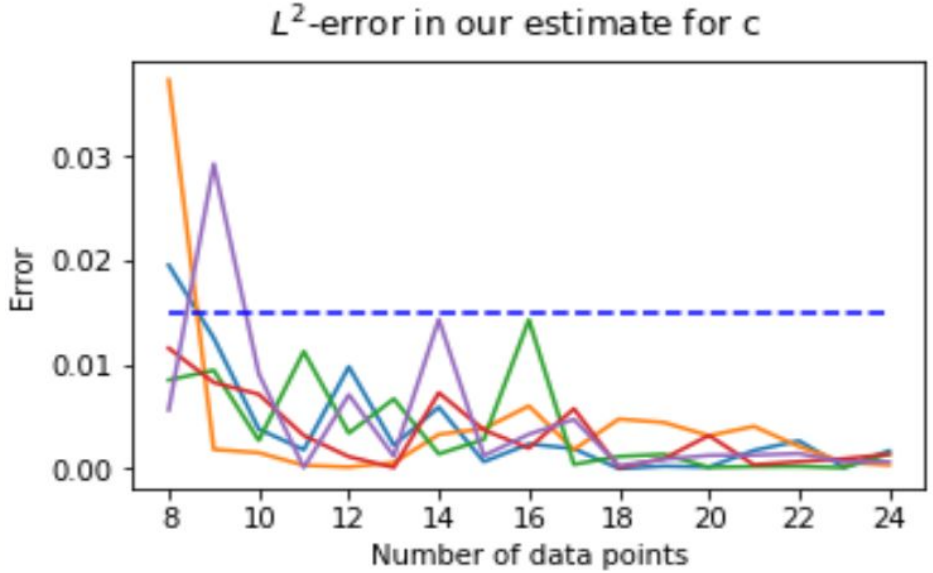
Here, the error is bounded by 0.041 for $10 \leq n \leq 24$ (blue-dashed line).

Calculating the L^2-error

- Given $\tilde{c}$, we can solve $\frac{d^2}{dt^2}\tilde{u}(x,t) - \tilde{c}\frac{d^2}{dx^2}\tilde{u}(x,t) = 0$ and get a solution based on our estimate:

$$\tilde{u}(x,t) = u(x, \sqrt{\tilde{c}}t) = (x - \sqrt{\tilde{c}}t)^2 + \sin(x + \sqrt{\tilde{c}}t)$$

- Can plot $\|\tilde{u} - u\|_{L^2}$ now:



The L^2-error is in our case bounded by 0.015 for $10 \leq n \leq 24$.

Contents

Burgers' Equation

$$\boxed{\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}}$$

We look at the *inviscid Burgers' Equation*, that is when the diffusion coefficient is zero: $\nu = 0$. Then a solution is:

$$u(x,t) = \frac{x}{1+t}$$

We implemented two similar setups:
1) Infer c in:

$$u_t + cuu_x = 0 \tag{1}$$

2) Infer ν in:

$$u_t + uu_x = \nu u_x \tag{2}$$

Applying our algorithm

- Used discretization methods (with step size $\tau = 0.001$)
- Replaced the non-linear term with the mean μ_{n-1} of u_{n-1}

- Used the backward Euler scheme for (1):

$$\frac{u_n(x) - u_{n-1}(x)}{\tau} + c u_n(x) \frac{d}{dx} u_n(x) = 0,$$

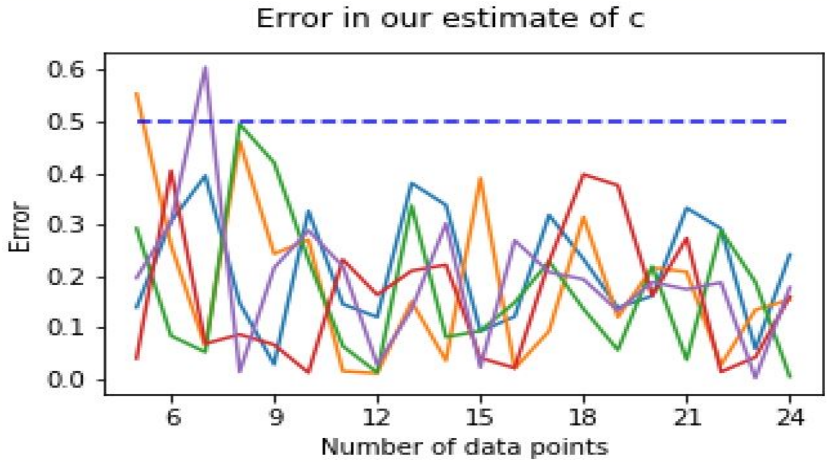
where $\mathcal{L}_x^c u_n = u_{n-1}$ and $\mathcal{L}_x^c = \cdot + \tau c \mu_{n-1} \frac{d}{dx} \cdot$.

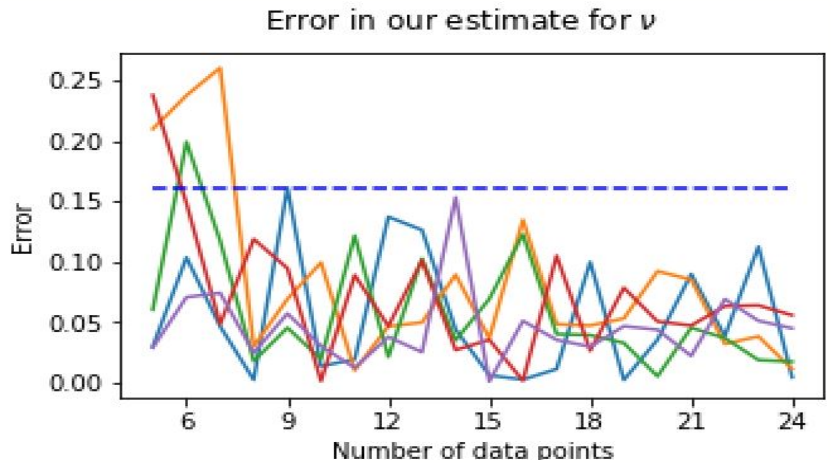
- Used the forward Euler scheme for (2):

$$\frac{u_n(x) - u_{n-1}(x)}{\tau} + u_{n-1}(x) \frac{d}{dx} u_{n-1}(x) = \nu \frac{d}{dx} u_{n-1}(x),$$

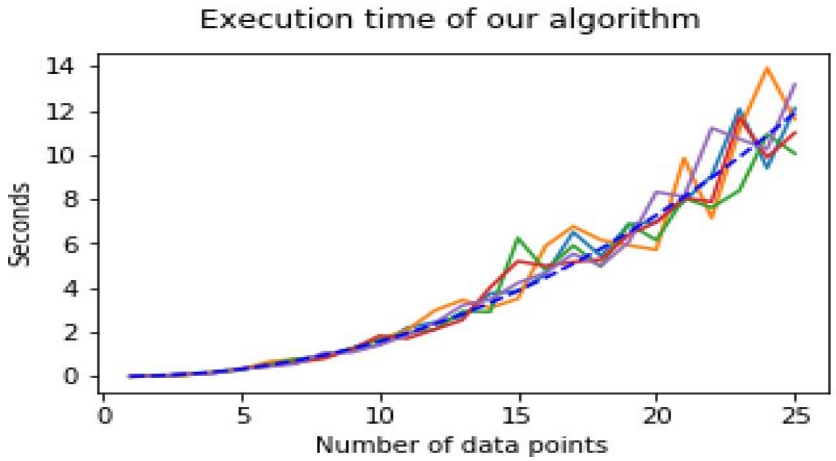
where $\mathcal{L}_x^\nu u_{n-1} = u_n$ and $\mathcal{L}_x^\nu = \cdot + \tau(\nu - \mu_{n-1}) \frac{d}{dx} \cdot$.

Results

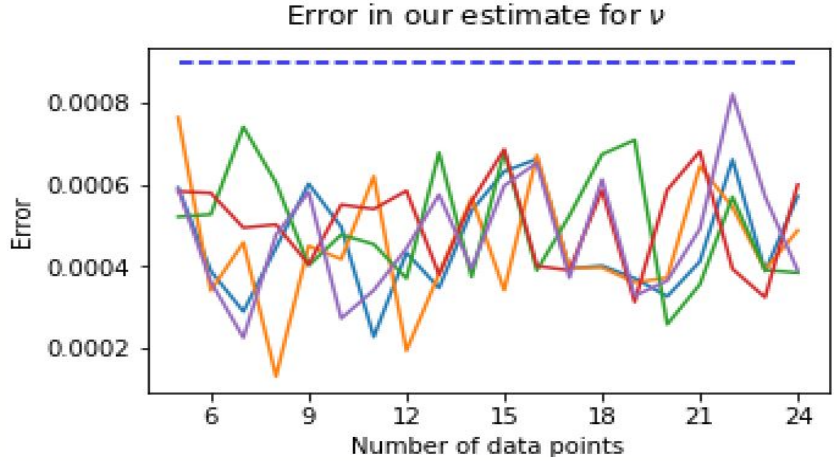




The blue-dashed line is given by $f(x) = 0.01x^{2.2}$:



Here, we replaced the non-linearity by $u(x, 0) = x$ for a comparison:



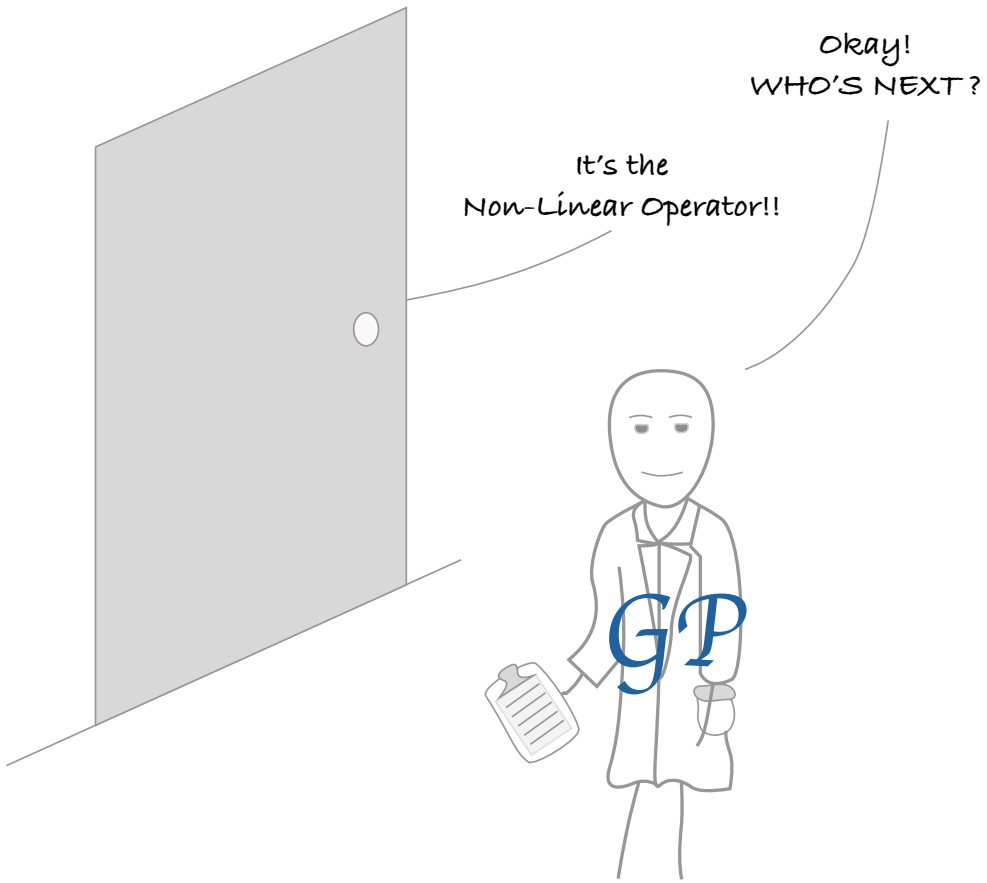
Contents

Problem solved!

Conclusions

- Efficient and quite accurate framework for estimating parameters in differential equations
- No discretization methods needed
- Designed for linear transformations only
- Covariance matrix often ill-conditioned for more than 30 data points
- Automatic calculation of all kernels possible
- Initial attempt with pyGPs was unsuccessful

What's next?



Oh yes!

Thank you!

Trying to use the Python-package pyGPs

Our pyGPs approach:
1. Assume Gaussian Priors with RBF Kernels:

$$u(x) \sim \mathcal{GP}(0, k_{uu}(x, x'; \sigma_u, l_u))$$

$$f(x) \sim \mathcal{GP}(0, k_{ff}(x, x'; \sigma_f, l_f))$$

2. Can optimize hyperparameters with pyGPs (given the data $\{X_u, Y_u\}$ and $\{X_f, Y_f\}$)
3. Know that the covariance matrix for f is $k_f = \mathcal{L}^{\phi}_{x'} \mathcal{L}^{\phi}_{x} k_{uu}$, since $f(x) = \mathcal{L}^{\phi}_{x} u(x)$. Set $k_f \simeq k_{ff}$. Then:

$$k_f(x_i, x_i) \simeq k_{ff}(x_i, x_i)$$

Rearranging:

$$\phi \simeq g(\sigma_u, \sigma_f, l_u),$$

This we can evaluate.

Using it for a simple example

We used this approach for $u(x) = \sqrt{x}$ and $f(x) = \mathcal{L}_x^\phi u(x)$ with $\phi = 12$. By the previous slide it follows

$$\phi \simeq \frac{\sigma_f}{\sigma_u}.$$

Using 15 evenly spaced data samples in $[0, 2\pi]$, our result was $\phi = 12.05$.