Fast Gaussian Process Methods for Point Process Intensity Estimation

John P. Cunningham, Maneesh Sahani, Krishna V. Shenoy

Stanford University

Gatsby, University College London

jcunnin@stanford.edu

Outline

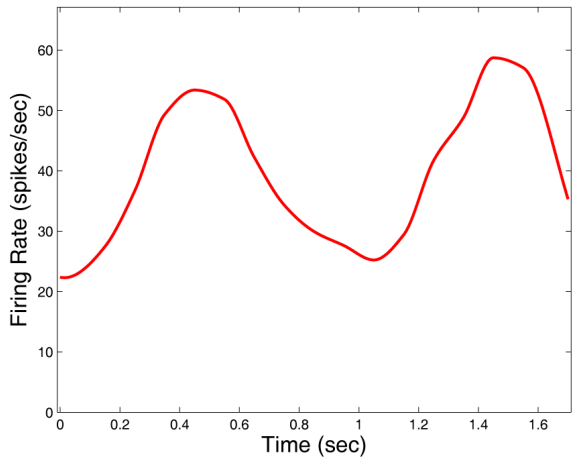
- Introduction
- Problem Statement
- Specific Implementation
- Algorithmic Solution
- Results
- Generalizing to other problems
- Conclusion

Outline

- Introduction
- Problem Statement
- Specific Implementation
- Algorithmic Solution
- Results
- Generalizing to other problems
- Conclusion

Introduction

True Intensity Function (Hidden)

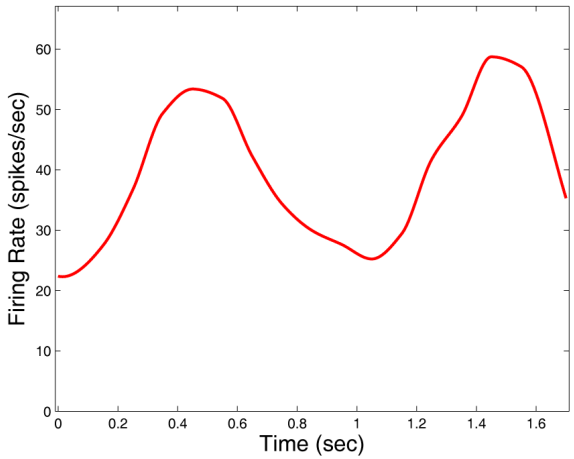


Noisy Point Process Data (Observed)

- Doubly-stochastic point processes (Cox processes)
- Used in finance, economics, neuroscience, ecology, etc.

Introduction

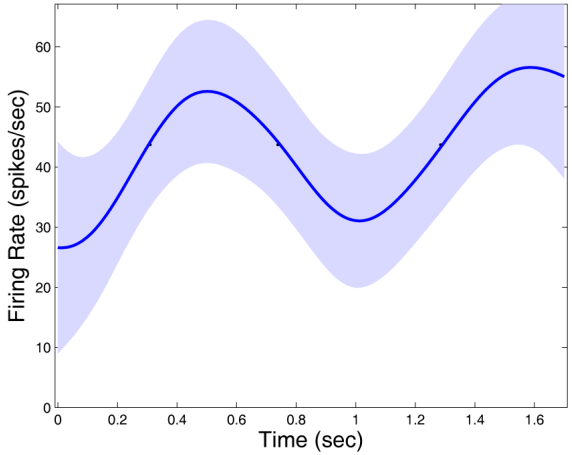
True Intensity Function (Hidden)



Noisy Point Process Data (Observed)

(Desired)

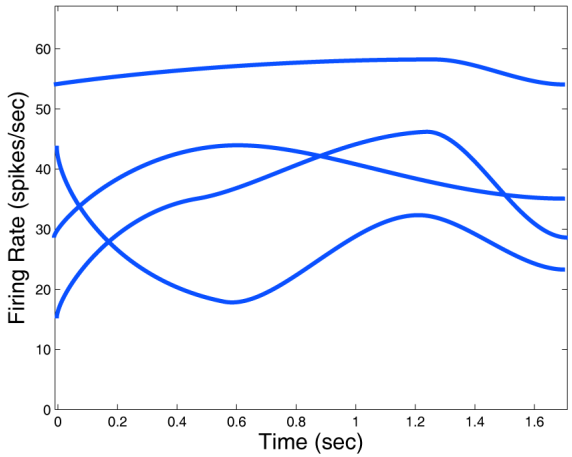
Estimate of Intensity Function



Introduction

Prior on Intensity Function

$p(x|\theta)$

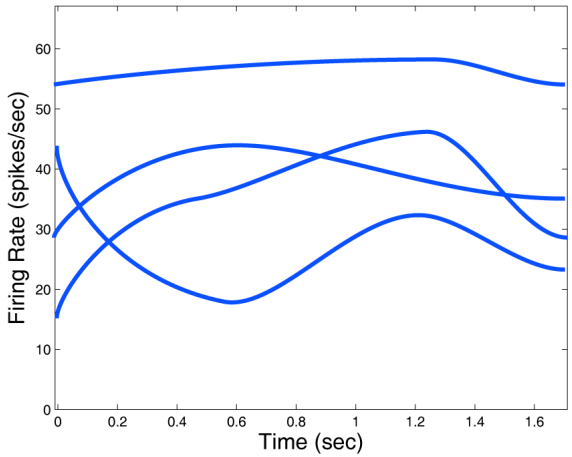


Gaussian Process

Introduction

Prior on Intensity Function

$p(x|\theta)$



Gaussian Process

Conditional Point Process Distribution

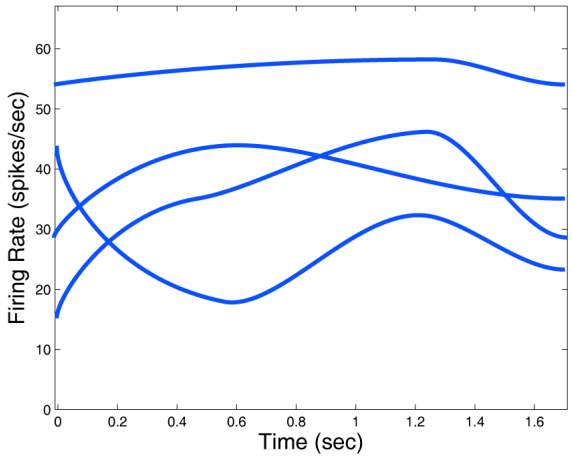
$p(y|x,\theta)$

(e.g. Inhomog. Gamma Interval Proc.)

Introduction

Prior on Intensity Function

$p(x|\theta)$



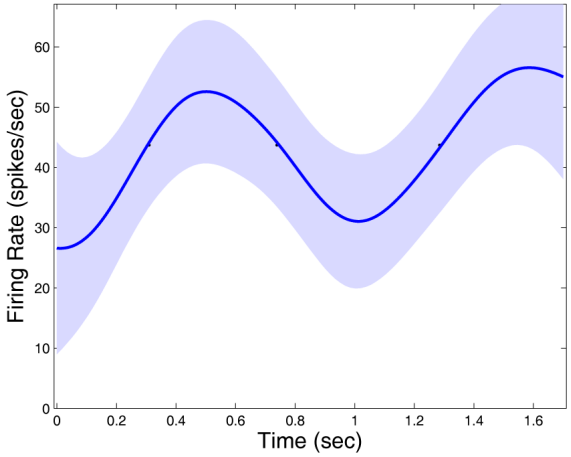
Gaussian Process

Conditional Point Process Distribution

$p(y|x,\theta)$

(e.g. Inhomog. Gamma Interval Proc.)

Estimate of Intensity Function



Model Selection

$\theta^* = \text{argmax } p(\theta|y)$

MAP Estimation

$x^* = \text{argmax } p(x|y,\theta^*)$

•Cunningham, Yu, Shenoy, Sahani (2008) Inferring neural firing rates from spike trains using Gaussian processes. *Advances in Neural Information Processing Systems (NIPS)* 20.

Outline

- Introduction
- Problem Statement
- Specific Implementation
- Algorithmic Solution
- Results
- Generalizing to other problems
- Conclusion

Problem Statement

- Computationally impractical (infeasible)
  - Run-time complexity is $O(n^3)$
  - Memory requirement is $O(n^2)$
  - n is large (thousands or more)
- How can we solve this problem?
  - Large scale optimization techniques
  - Problem specific algorithmic manipulations
- Does these methods generalize?
  - Optimization and Gaussian Process ‘bag of tricks’

Specific Implementation

- Gaussian Process (GP) prior

$$p(\mathbf{x} \mid \theta) = \mathcal{N}(\mu\mathbf{1}, \Sigma)$$

  - (covariance parameterized by a kernel such as SE)

$$\Sigma = \left\{K(t_i, t_j)\right\}_{i,j\in\{1,\ldots,n\}} \text{ where } K(t_i, t_j) = \sigma_f^2 \exp\left\{-\frac{\kappa}{2}(t_i - t_j)^2\right\} + \sigma_v^2 \delta_{ij}$$

- Log-concave renewal processes
  - (interval primitive, intensity rescaling, discretization)
  - Here: Inhomogeneous Gamma Interval Process

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{N}\left[\frac{\gamma x_{y_i}}{\Gamma(\gamma)}\left(\gamma \sum_{k=y_{i-1}}^{y_i-1} x_k \Delta\right)^{\gamma-1} \exp\left\{-\gamma \sum_{k=y_{i-1}}^{y_i-1} x_k \Delta\right\}\right]$$

Specific Implementation

- Model Selection with a Laplace approximation

$$\begin{aligned}\hat{\theta} &= \underset{\theta}{\operatorname{argmax}}\, p(\theta)p(\mathbf{y} \mid \theta) \\ &\approx \underset{\theta}{\operatorname{argmax}}\, p(\theta)p(\mathbf{y} \mid \mathbf{x}^*, \theta)p(\mathbf{x}^* \mid \theta)\frac{(2\pi)^{\frac{n}{2}}}{|\Lambda^* + \Sigma^{-1}|^{\frac{1}{2}}},\end{aligned}$$

- where $\Lambda^* = -\nabla^2_{\mathbf{x}} \log\, p(\mathbf{y} \mid \mathbf{x}, \theta) \mid_{\mathbf{x}=\mathbf{x}^*}$

- MAP estimation (to find modal x* at any θ) with a log barrier Newton Method

$$\mathbf{x}^* = \underset{\mathbf{x} \succeq \mathbf{0}}{\operatorname{argmax}}\, p(\mathbf{x} \mid \mathbf{y}) = \underset{\mathbf{x} \succeq \mathbf{0}}{\operatorname{argmax}}\, p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})$$

Specific Implementation

- Computational bottlenecks include:
  - MAP estimation
    - Objective $f(\mathbf{x}) = -\log p(\mathbf{y} \mid \mathbf{x}, \theta) p(\mathbf{x} \mid \theta)$
    - Objective gradients $\mathbf{g} = \nabla_{\mathbf{x}} f$
    - Newton steps $\mathbf{x}_{nt} = -H^{-1}\mathbf{g}$ where $H = \nabla^2_{\mathbf{x}} f_\tau(\mathbf{x}) = \Sigma^{-1} + \Lambda$
  - Model evidence and its gradients (the Laplace approximation)

$$-\log\ p(\mathbf{y} \mid \theta) \approx -\log\ p(\mathbf{y} \mid \mathbf{x}^*)\ + \frac{1}{2}(\mathbf{x}^* - \mu\mathbf{1})^T \Sigma^{-1}(\mathbf{x}^* - \mu\mathbf{1})\ + \frac{1}{2}\log|I + \Sigma\Lambda^*|$$

- We discuss methods to reduce run time and memory requirements drastically (without loss of accuracy)

Outline

- Introduction
- Problem Statement
- Specific Implementation
- Algorithmic Solution
- Results
- Generalizing to other problems
- Conclusion

Algorithmic Solution (1/3) – MAP Estimation

- We must calculate the Newton step ($\mathbf{x}_{nt} = -H^{-1}\mathbf{g}$), where $-H^{-1} = -(\Sigma^{-1} + \Lambda)^{-1}$
- Two costly O(n^3) inversions per step and O(n^2) memory
- We show that a decomposition $\Lambda = RR^T$ can be found in closed form, allowing us to write:

$$\begin{aligned} -H^{-1} &= -(\Sigma^{-1} + \Lambda)^{-1} \\ &= -\Sigma + \Sigma R(I + R^T \Sigma R)^{-1} R^T \Sigma \end{aligned}$$

- This prevents costly matrix inversions, making the complexity that of solving $(I + R^T \Sigma R)^{-1}\mathbf{v}$ where $\mathbf{v} = R^T \Sigma \mathbf{g}$
- This is quickly done via conjugate gradients and fast multiplication methods (linear in R, FFT for Σ)

Algorithmic Solution (2/3) – MAP Estimation

- Note recursion in Newton's Method (each iteration step size $t^{(j)}$):

$$\begin{aligned}(\mathbf{x}^{(k)} - \mu\mathbf{1}) &= \mathbf{x}^{(k-1)} + t^{(k-1)}\mathbf{x}_{nt}^{(k-1)} - \mu\mathbf{1} \\ &= \sum_{j=1}^{k-1} t^{(j)}\mathbf{x}_{nt}^{(j)} + (\mathbf{x}^{(0)} - \mu\mathbf{1})\end{aligned}$$

- Using the previous form (matrix inv. lemma) of the Hessian (and, as such, the Newton step $\mathbf{x}_{nt} = -H^{-1}\mathbf{g}$), we write:

$$\Sigma^{-1}(\mathbf{x}^{(k)} - \mu\mathbf{1}) =$$

$$\sum_{j=1}^{k-1} t^{(j)}\left(-\mathbf{g}^{(j)} + R^{(j)}(I + R^{(j)T}\Sigma R^{(j)})^{-1}R^{(j)T}\Sigma\mathbf{g}^{(j)}\right)$$

- Now neither the objective nor the gradient has any matrix inversions; in fact, we get these terms 'for free' (only linear operations such as inner products) from the Newton step

Algorithmic Solution (3/3) – Model Selection

- Evidence (marginal likelihood) approximation

$$-\log\ p(\mathbf{y} \mid \theta) \approx -\log\ p(\mathbf{y} \mid \mathbf{x}^*)\ +\ \frac{1}{2}(\mathbf{x}^* - \mu\mathbf{1})^T\Sigma^{-1}(\mathbf{x}^* - \mu\mathbf{1})\ +\ \frac{1}{2}\log|I + \Sigma\Lambda^*|$$

  - First two terms of RHS (and gradients) are already calculated
  - Consideration required for third term: $\log|I + \Sigma\Lambda^*|$

- Recall special structure: we decompose $\Lambda^* = USU^T$

- There are only m (<<n) meaningful (near axis-aligned) eigenvalues, allowing us to approximate $O(n^3)$ computations with $O(m^3)$

$$\begin{aligned}\log|I + \Sigma\Lambda^*| &= \log|I + \Sigma USU^T| \\ &= \log|I + U^T\Sigma US| \\ &\approx \log|I + \overline{\Sigma}\,\overline{S}|\end{aligned}$$

Outline

- Introduction
- Problem Statement
- Specific Implementation
- Algorithmic Solution
- Results
- Generalizing to other problems
- Conclusion

Results

- Our algorithmic solution should have:
  - Large run time improvement: $O(n^3)$ becomes (roughly) O(nlogn)
  - Memory burden eliminated: $O(n^2)$ becomes O(n)
  - Effectively no loss in accuracy
- To test these claims, we:
  - Pick representative intensity functions over various lengths of time
  - Generate point process data from these intensities
  - Calculate times and accuracies of the MAP estimations
  - Calculate times and accuracies of the evidence calculations
  - Calculate times and accuracies of the full iterative method
- These calculations are done for both the Fast algorithm and a Naive method (typical MATLAB/Linux setup)

Results

	Data Set					
	1	2	3	4	5	6
Data Size(n)	500	1000	1000	2000	4000	10000
Num. Events (N)[1]	20-30	30-40	140-160	55-70	55-70	140-160
MAP Estimation						
Fast Solve Time(s)	0.12	0.17	0.46	0.32	7.6	37.9
Naive Solve Time(s)	7.04	40.5	39.5	333	3704	1day[3]
Speed Up	**58**×	**232**×	**86**×	**1043**×	**493**×	**2000**×[3]
MS Error (Fast vs. Naive)[2]	4.3e-4	4.2e-4	2.1e-4	5.2e-6	6.1e-6	-
Avg. CG Iters.	6.4	5.5	16.2	8.1	29.9	49.7
Log Determinant Approximation						
Fast Solve Time(s)	6.5e-4	1.8e-3	1.9e-2	2.8e-3	2.8e-3	2.5e-2
Naive Solve Time(s)	0.24	1.02	0.97	5.7	34.7	540[3]
Speed Up	**375**×	**566**×	**52**×	**2058**×	**1.3e4**×	**2.2e4**×[3]
Avg. Acc. of Fast Approx.	99.1%	98.8%	99.8%	98.9%	99.7%	-
Avg. Model Selection Iters.	54.3	54.6	89.1	68.1	39.4	40.7
Full GP Intensity Estimation (Iterative Model Selection and MAP Estimation)						
Fast Solve Time(s)	4.4	7.1	30.3	18.7	128	423
Naive Solve Time(s)	443	3094	4548	2.4e4	1.5e5	1month[3]
Speed Up	**105**×	**451**×	**150**×	**1512**×	**1166**×	**1e4**×[3]
MS Error (Fast vs. Naive)[2]	0.10	0.03	10.8	0.01	0.01	-

[1] Entries show a range of data used.

[2] Squared norm of $x(t)$ is roughly 10^3 to 10^5, so these errors are insignificant.

[3] Unable to complete naive method; numbers estimated from cubic scaling.

Outline

- Introduction
- Problem Statement
- Specific Implementation
- Algorithmic Solution
- Results
- Generalizing to other problems
- Conclusion

Generalizing this result

- We've discussed a very specific problem implementation, but the methods are general
- Many GP regression problems have similar structure (e.g. the Hessian or the Laplace approximation)

$$H = \nabla^2_{\mathbf{x}} f_\tau(\mathbf{x}) = \Sigma^{-1} + \Lambda$$

$$-\log\ p(\mathbf{y} \mid \theta) \approx -\log\ p(\mathbf{y} \mid \mathbf{x}^*) + \frac{1}{2}(\mathbf{x}^* - \mu\mathbf{1})^T \Sigma^{-1}(\mathbf{x}^* - \mu\mathbf{1}) + \frac{1}{2}\log|I + \Sigma\Lambda^*|$$

- Large scale optimization 'bag of tricks':
  - (Preconditioned) conjugate gradients (PCG)
  - Implicit linear operations (avoids matrices in memory)
  - Fast matrix multiplication methods (FFT, multipole, etc.)
  - Decomposing matrices with special structure and using matrix inversion lemma, Sylvester's determinant rule, etc.
  - Exploiting recursions to avoid unnecessary computation

Conclusion

- Orders of magnitude run-time improvement can be found by careful problem inspection
- Memory burden can be completely eliminated by avoiding explicit matrix representations
- This ‘bag of tricks’ is general and powerful
- Acknowledgements
  - Support
    - NIH-NINDS-CRCNS-R01, Michael Flynn SGF, NSF, Gatsby, CDRF, BWF, ONR, Sloan, and Whitaker
  - Collaborators
    - Stephen Boyd (Electrical Engineering, Stanford)
  - Staff
    - Drew Haven
    - Sandy Eisensee