Assignment 9

Reid Ginoza

4/12/2020

Analytical Maximum Likelihood

This is Exercise 11.1 from (Särkkä and Solin 2019). We are given the following:

$$x_k = \theta + r_k, \quad k = 1, 2, \ldots, T, \tag{1}$$

where $r_k \sim \mathrm{N}\left(0, \sigma^2\right)$ are independent.

Maximum Likelihood Estimator

Let $L(\cdot)$ be the likelihood and $l(\cdot)$ be the negative log likelihood. Then, to find the maximum likelihood estimator $\theta_{\mathrm{ML}} = \arg\min_\theta L(\theta)$, we will minimize $l(\theta)$:

$$\begin{aligned}
x_k = \theta + r_k &\sim \mathrm{N}\left(\theta, \sigma^2\right) && (2)\\
\implies \mathrm{f}\left(x_k|\theta\right) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(x_k-\theta\right)^2}{2\sigma^2}\right) && (3)\\
\implies L(\theta) &= \left(2\pi\sigma^2\right)^{\frac{T}{2}} \prod_{k=1}^{T} \exp\left(-\frac{\left(x_k-\theta\right)^2}{2\sigma^2}\right) && (4)\\
&= \left(2\pi\sigma^2\right)^{\frac{T}{2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{k=1}^{T}\left(x_k-\theta\right)^2\right) && (5)\\
\implies l(\theta) &= -\frac{T}{2}\log\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}\sum_{k=1}^{T}\left(x_k-\theta\right)^2 && (6)\\
\text{Now, suppose } \theta_{\mathrm{ML}} \text{ solves the following,} \quad \frac{\mathrm{d}l}{\mathrm{d}\theta} &= 0 && (7)\\
&= -\frac{1}{\sigma^2}\sum_{k=1}^{T}\left(x_k-\theta_{\mathrm{ML}}\right) && (8)\\
\implies 0 &= \sum_{k=1}^{T}\left(x_k-\theta_{\mathrm{ML}}\right) && (9)\\
\implies \theta_{\mathrm{ML}} &= \frac{1}{T}\sum_{k=1}^{T} x_k = \bar{x} && (10)
\end{aligned}$$

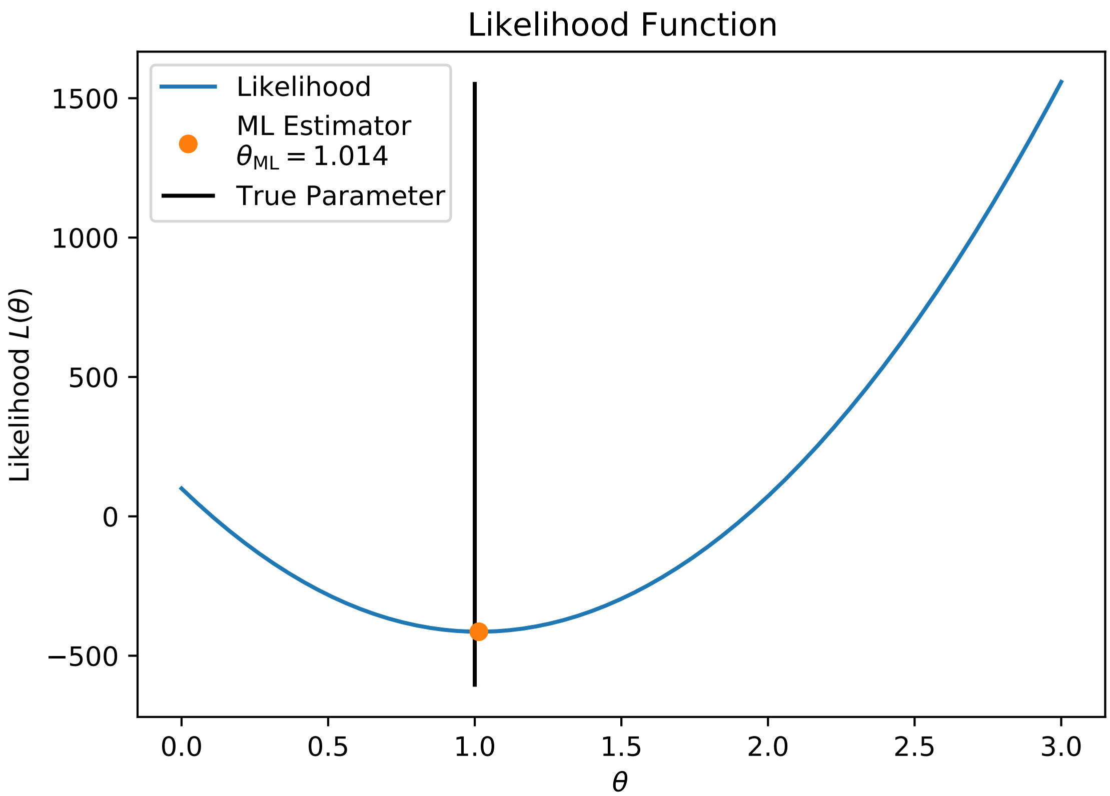
Numerical Maximum Likelihood

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import minimize_scalar
from scipy.stats import norm
```

```python
# -- Plotting Tools --
def low_buff(y_values):
    span = y_values.max() - y_values.min()
    return y_values.min() - 0.1 * span


# -- True Distribution --
THETA = 1
SIGMA = 1

# -- Samples
T = 1000

r = np.random.normal(scale=SIGMA, size=(T,))
sample = THETA + r
print(sample.mean())
```

```
## 1.0138583286773388
```

```python
def neg_log_llh(theta, sample, T, sigma):
    return - T/2 * np.log(2 * np.pi * sigma**2) + 1 / (2 * sigma) * ((sample - theta)**2).sum()


def nll_vec(theta_plot, sample, T, sigma):
    return np.array([neg_log_llh(theta, sample, T, sigma) for theta in theta_plot])


theta_plot = np.linspace(0, 3, 1000)
likelihood_plot = nll_vec(theta_plot, sample, T, SIGMA)
estimator_result = minimize_scalar(neg_log_llh, args=(sample, T, SIGMA))

fig, ax = plt.subplots()
ax.plot(theta_plot, likelihood_plot,
        label='Likelihood')
ax.vlines(x=THETA, ymin=low_buff(likelihood_plot), ymax=likelihood_plot.max(),
          label='True Parameter')
ax.plot(sample.mean(), neg_log_llh(sample.mean(), sample, T, SIGMA), 'o',
          label='ML Estimator\n' + r'$\theta_{\mathrm{ML}} = ' + str(round(sample.mean(), 3)) + '$')
# ax.plot(estimator_result.x, estimator_result.fun, 'o',
#         label=f'Numerical Minimization: {round(estimator_result.x, 3)},'
#               f' {round(estimator_result.fun, 3)}')
ax.set_xlabel(r'$\theta$')
ax.set_ylabel('Likelihood ' + r'$L(\theta)$')
ax.set_title('Likelihood Function')
ax.legend()
plt.show()
```



Analytical Maximum A Posteriori

This is Exercise 11.2 from (Särkkä and Solin 2019). Since this is not a nonlinear SDE, we can use the following to determine the posterior probability distribution:

$$\begin{aligned} p\left(\theta \mid x\left(t_1\right), \ldots, x\left(t_T\right)\right) \sim p\left(\theta\right) \exp\left(-l\left(\theta\right)\right) = \exp\left(-l_p\left(\theta\right)\right) \quad (11) \\ = \left[\frac{1}{\sqrt{2\pi\lambda^2}} \exp\left(-\frac{\theta^2}{2\lambda^2}\right)\right]\left[\exp\left(\frac{T}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\sum_{k=1}^{T}\left(x_k - \theta\right)^2\right)\right] \quad (12) \\ = \left(2\pi\lambda^2\right)^{\frac{1}{2}} \exp\left(-\frac{\theta^2}{2\lambda^2} + \frac{T}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\sum_{k=1}^{T}\left(x_k - \theta\right)^2\right) \quad (13) \end{aligned}$$

and to find the maximimum a posteriori estimate, $\theta_{\mathrm{MAP}} = \arg\min_\theta p\left(\theta \mid x\left(t_1\right), \ldots, x\left(t_T\right)\right)$, we minimize the negative logarithm of the posterior distribution, which we'll call A.

$$A = -\log\left(\left(2\pi\lambda^2\right)^{\frac{1}{2}} \exp\left(-\frac{\theta^2}{2\lambda^2} + \frac{T}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\sum_{k=1}^{T}\left(x_k - \theta\right)^2\right)\right) \tag{14}$$

$$= -\frac{1}{2}\log\left(2\pi\lambda^2\right) + \frac{\theta^2}{2\lambda^2} - \frac{T}{2}\log\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}\sum_{k=1}^{T}\left(x_k - \theta\right)^2 \tag{15}$$

$$= -\frac{1}{2}\log\left(2\pi\lambda^2\right) + \frac{\theta^2}{2\lambda^2} - \frac{T}{2}\log\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}\sum_{k=1}^{T}x_k^2 - 2\theta\frac{1}{2\sigma^2}\sum_{k=1}^{T}x_k + \frac{1}{2\sigma^2}T\theta^2 \tag{16}$$

$$= \left[\frac{1}{2\lambda^2} + \frac{T}{2\sigma^2}\right]\theta^2 - \left[\frac{1}{\sigma^2}\sum_{k=1}^{T}x_k\right]\theta + \left[\frac{1}{2\sigma^2}\sum_{k=1}^{T}x_k^2 - \frac{1}{2}\log\left(2\pi\lambda^2\right)\right] \tag{17}$$

Since A is quadratic in θ, we can use the familiar formula for the minimum.

$$\theta_{\text{MAP}} = \frac{\frac{1}{\sigma^2}\sum_{k=1}^{T}x_k}{\frac{1}{\lambda^2} + \frac{T}{\sigma^2}} \tag{18}$$

$$= \frac{\frac{1}{\sigma^2}\sum_{k=1}^{T}x_k}{\frac{T\lambda^2 + \sigma^2}{\lambda^2\sigma^2}} \tag{19}$$

$$= \frac{\lambda^2}{T\lambda^2 + \sigma^2}\sum_{k=1}^{T}x_k \tag{20}$$

$$= \frac{1}{T + \frac{\sigma^2}{\lambda^2}}\sum_{k=1}^{T}x_k \tag{21}$$

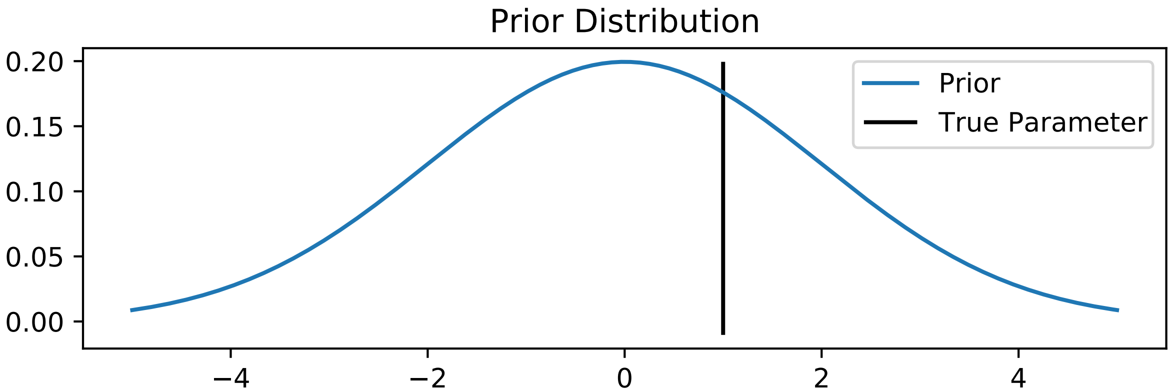
Numerical MAP Estimator

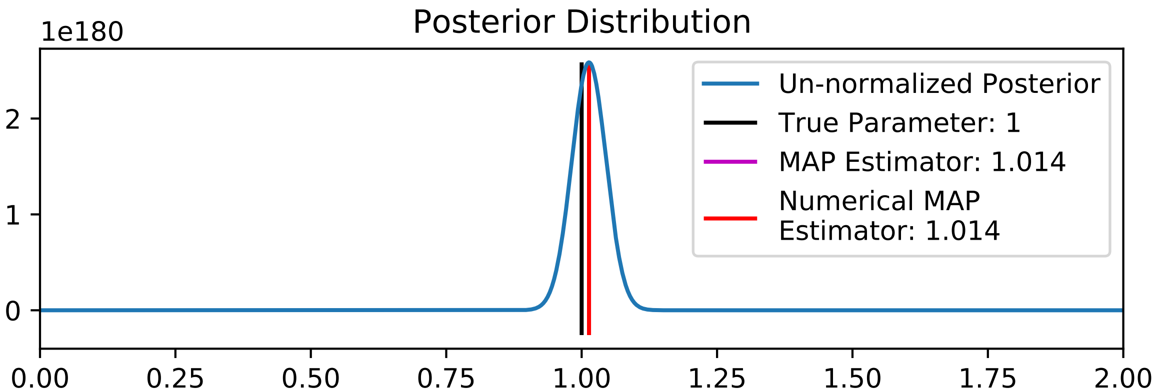
```python
# -- Prior --
LAMBDA = 2


def prior(theta, lamb):
    return 1 / np.sqrt(2 * np.pi * lamb**2) * np.exp(- theta**2 / (2 * lamb**2))


def posterior(theta, lamb, sigma, sample):
    """ f1 * exp(f2 + f3 + f4) """
    f1 = np.sqrt(2 * np.pi * lamb**2)
    f2 = - theta**2 / (2 * lamb**2)
    f3 = sample.size / 2 * np.log(2 * np.pi * sigma**2)
    f4 = - 1 / (2 * sigma**2 ) * np.sum((sample - theta)**2)
    return f1 * np.exp(f2 + f3 + f4)


def map_estimate(T, sigma, lamb, sample):
```

```python
    """ Uses analytical formula found by minimizing the
    un-normalized negative log of the posterior """
    return 1 / (T + sigma**2/lamb**2) * sample.sum()


def this_neg_posterior(theta):
    """ dependent on current samples
    convenience function for optimizer of the posterior
    without mixing arrays (i.e. sample size vs attempted theta) """
    return -1 * posterior(theta, LAMBDA, SIGMA, sample)


# Find Estimators
num_map_estimate = minimize_scalar(this_neg_posterior)
theta_map = map_estimate(T, SIGMA, LAMBDA, sample)

# plotting
theta_plot = np.linspace(-5, 5, 5000)
prior_plot = prior(theta_plot, LAMBDA)
posterior_plot = np.array([posterior(th, LAMBDA, SIGMA, sample) for th in theta_plot])

fig2, axes2 = plt.subplots(nrows=2)
# prior plot
axes2[0].plot(theta_plot, prior_plot, label='Prior')
axes2[0].vlines(x=THETA, ymin=low_buff(prior_plot), ymax=prior_plot.max(),
          label='True Parameter')
axes2[0].set_title('Prior Distribution')
axes2[0].legend()

# posterior plot
# Doesn't make sense to show prior since posterior is un-normalized
# axes2[1].plot(theta_plot, prior_plot, label='Prior', ls='dashed', alpha=0.8)
axes2[1].plot(theta_plot, posterior_plot, label='Un-normalized Posterior')
axes2[1].vlines(x=THETA, ymin=low_buff(posterior_plot), ymax=posterior_plot.max(),
          label=f'True Parameter: {THETA}')
axes2[1].vlines(x=theta_map, ymin=low_buff(posterior_plot), ymax=posterior_plot.max(),
          colors='m', label=f'MAP Estimator: {round(theta_map, 3)}')
axes2[1].vlines(x=num_map_estimate.x, ymin=low_buff(posterior_plot), ymax=posterior_plot.max(),
          colors='r', label=f'Numerical MAP\nEstimator: {round(num_map_estimate.x, 3)}')
axes2[1].set_xlim(left=0, right=2)
```

```
## (0.0, 2.0)
```

```python
axes2[1].set_title('Posterior Distribution')
axes2[1].legend()
fig2.tight_layout()
plt.show()
```





Metropolis-Hastings Algorithm

This is Exercise 11.3 from (Särkkä and Solin 2019). This is strictly a coding exercise, using the Metropolis-Hastings algorithm to approximate the posterior distribution.

```
# Functions for M-H Algorithm
def sample_proposal(cur_theta, gamma):
    return np.random.normal(loc=cur_theta, scale=gamma)


def proposal_probability(prop_theta, loc, gamma):
    return norm(loc=loc, scale=gamma).pdf(prop_theta)


def l_p(theta, lamb, sigma, sample):
    """ Notes on derivation
    l(th) is the negative log-likelihood
    p(th) is the prior distribution

    l_p = l(th) - log(p(th))
        = - log( p(th) * exp(-l(th) )

    written in the form f1*theta**2 + f2 * theta + f3 + f4
    """
```

```python
    T = sample.size
    f1 = 1 / (2 * lamb**2) + T / (2 * sigma**2)
    f2 = - 1 / sigma**2 * (sample.sum())
    f3 = 1 / (2 * sigma**2) * np.sum((sample**2))
    f4 = - 1/2 * np.log(2 * np.pi * lamb**2)
    return f1 * theta**2 + f2 * theta + f3 + f4


def acceptance_probability(cur_theta, prop_theta, lamb, sigma, sample, gamma):
    ratio_1 = np.exp(l_p(cur_theta, lamb, sigma, sample)
                     - l_p(prop_theta, lamb, sigma, sample))
    ratio_2 = (proposal_probability(cur_theta, prop_theta, gamma)
               / proposal_probability(prop_theta, cur_theta, gamma))
    return min(1, ratio_1 * ratio_2)


def metropolis_hastings(theta_0, gamma, lamb, sigma, sample, num_steps):
    mcmc_sampling = [theta_0]
    cur_theta = theta_0
    accept_count = 0
    for _ in range(num_steps):
        prop_theta = sample_proposal(cur_theta, gamma)
        alpha = acceptance_probability(
            cur_theta, prop_theta,lamb, sigma, sample, gamma
        )
        u = np.random.uniform()
        if u < alpha:  # accept
            mcmc_sampling.append(prop_theta)
            cur_theta = prop_theta
            accept_count += 1
        else:
            mcmc_sampling.append(cur_theta)
    acceptance_rate = accept_count / num_steps
    print(f'Acceptance Probability: {acceptance_rate}')
    return mcmc_sampling, acceptance_rate


# Metropolis-Hastings Values
MCMC_TRIALS = 10000
THETA_0 = 0
GAMMA = .15

mh_sampling, acceptance_rate = metropolis_hastings(
    THETA_0, GAMMA, LAMBDA, SIGMA, sample, MCMC_TRIALS
)

# construct analytical posterior normalized:
```

```
## Acceptance Probability: 0.2531
```

```python
mh_theta_plot = np.linspace(
    np.min(mh_sampling),
    np.max(mh_sampling),
```

```
    1000,
)
mh_analytical_post = np.array([posterior(th, LAMBDA, SIGMA, sample) for th in mh_theta_plot])
tmp_hist = np.histogram(mh_sampling, 'auto')
peak = tmp_hist[0].max()
del tmp_hist
mh_analytical_post *= peak / mh_analytical_post.max()

# Plot mh_sampling over time
mh_fig, mh_axes = plt.subplots(nrows=2)
mh_axes[0].plot(mh_sampling)
mh_axes[0].set_xlabel('Iteration')
mh_axes[0].set_ylabel(r'$\theta$')
mh_axes[0].set_title(f'Sampling History\nAcceptance Rate: {round(acceptance_rate, 3)}')

# Plot distribution
_ = mh_axes[1].hist(mh_sampling, bins='auto', label='MCMC')
mh_axes[1].plot(mh_theta_plot, mh_analytical_post, label='Analytical', ls='dashed')
```

```
## [<matplotlib.lines.Line2D object at 0x120461880>]
```

```
mh_axes[1].vlines(x=THETA, ymin=0, ymax=mh_analytical_post.max(),
                  label=r'True Parameter: $\theta =$' + f' {THETA}')
```

```
## <matplotlib.collections.LineCollection object at 0x11dfe1640>
```

```
mh_axes[1].set_xlabel(r'$\theta$')
```

```
## Text(0.5, 0, '$\\theta$')
```

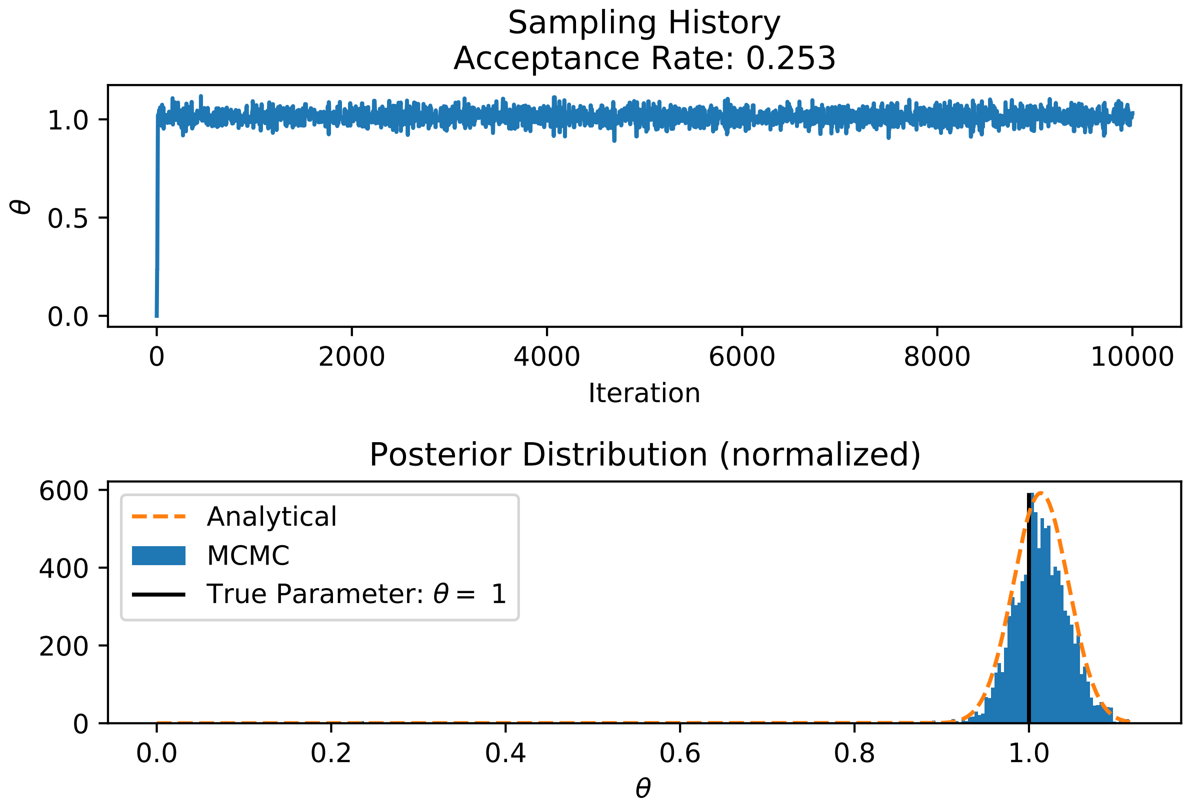
```
mh_axes[1].set_title(r'Posterior Distribution (normalized)')
```

```
## Text(0.5, 1.0, 'Posterior Distribution (normalized)')
```

```
mh_axes[1].legend()
```

```
## <matplotlib.legend.Legend object at 0x11dfe1f40>
```

```
mh_fig.tight_layout()
plt.show()
```



References

Särkkä, Simo, and Arno Solin. 2019. *Applied Stochastic Differential Equations*. Vol. 10. Cambridge University Press.