Aula5_1_Modelos_de_espaço_de_estado

August 13, 2020

1 Modelos de espaço de estado

- Modelos estruturais
- Previsão Bayesiana

por **Cibele Russo**

Baseado em

- Commandeur, Jacques J. F., and Siem Jan Koopman. 2007. An Introduction to State Space Time Series Analysis. Oxford ; New York: Oxford University Press.
- Durbin, James, and Siem Jan Koopman. 2012. Time Series Analysis by State Space Methods: Second Edition. Oxford University Press.
- Franco, G. C., Gamerman, D., & Santos, T. R. (2009). Modelos de espaço de estados: abordagens clássica e bayesiana. São Paulo: Associação Brasileira de Estatística. Disponível em ftp://www.est.ufmg.br/pub/glaura/Series%20Temporais/MEE/minicurso-MEE-ESTE.pdf. Acessado em 01/08/2020.

Leituras adicionais:

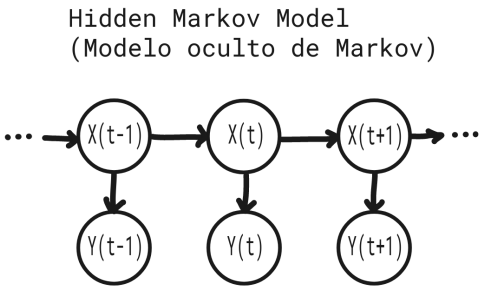
- https://www.statsmodels.org/stable/statespace.html
- https://www.statsmodels.org/devel/examples/notebooks/generated/statespace_local_linear_trend.htm
- http://www.chadfulton.com/fulton_statsmodels_2017/sections/1-introduction.html
- https://faculty.chicagobooth.edu/john.cochrane/research/papers/time_series_book.pdf

1.1 Modelos de espaço de estado

- Bastante utilizados em engenharia de controle
- Na abordagem clássica, são usados "modelos estruturais"
- Na abordagem bayesiana, são usados "modelos dinâmicos"
- Permitem a modelagem multivariada de dados
- Permitem a inclusão de variáveis endógenas e exógenas
- Alguns exemplos de aplicação:

– Educação: Prever o nível educacional, com variáveis exógenas, por exemplo acesso a escola, renda, e endógenas, por exemplo habilidades individuais

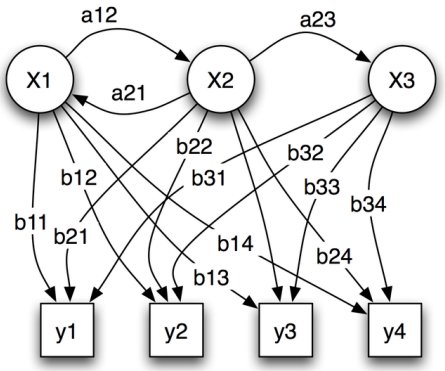
– Finanças: Prever o valor de uma ação, com variáveis exógenas, por exemplo, política externa, e endógenas, por exemplo, lucro da empresa



A variável oculta $X(t)$ (no instante de tempo t depende exclusivamente do valor da variável escondida $X(t-1)$ (no instante de tempo $t-1$, o que é chamado de propriedade de Markov. Da mesma forma, o valor da variável observada $Y(t)$ depende exclusivamente do valor da variável escondida $X(t)$ (ambas no instante de tempo t).

Exemplo de parâmetros probabilísticos de um modelo oculto de Markov

- x: estados
- y: possíveis observações
- a: probabilidades de transição de estado
- b: probabilidades de saídas



Fonte: https://pt.wikipedia.org/wiki/Modelo_oculto_de_Markov

1.2 Análise de séries temporais via método de espaço de estado

Fonte: https://www.statsmodels.org/stable/statespace.html

Um modelo geral de espaço de estado pode ser escrito na forma

$$\begin{aligned} y_t &= Z_t \alpha_t + d_t + \varepsilon_t \\ \alpha_{t+1} &= T_t \alpha_t + c_t + R_t \eta_t \end{aligned}$$

em que

- y_t se refere ao vetor observado no tempo t,
- α_t se refere ao vetor de estado no tempo t,

e em que os componentes aleatórias são definidos como

$$\begin{aligned} \varepsilon_t &\sim N(0, H_t) \\ \eta_t &\sim N(0, Q_t) \end{aligned}$$

As variáveis restantes $(Z_t, d_t, H_t, T_t, c_t, R_t, Q_t)$ nas equações são matrizes que descrevem o processo. Seus nomes e dimensões são dadas por

- Z: matriz de desenho $(k\_endog \times k\_states \times nobs)$
- d: intercepto das observações $(k\_endog \times nobs)$
- T: matriz de transição $(k\_states \times k\_states \times nobs)$
- c: intercepto do estado $(k\_states \times nobs)$
- R: matriz de seleção $(k\_states \times k\_posdef \times nobs)$
- H: matriz de covariância das observações $(k\_endog \times k\_endog \times nobs)$
- Q: covariância de estado $(k\_posdef \times k\_posdef \times nobs)$

No caso em que uma das matrizes seja invariante no tempo (por exemplo $Z_t = Z_{t+1} \ \forall \ t$), sua última dimensão seria $nobs = 1$.

Esta forma genérica encapsula muitas das séries temporais lineares mais populares modelos (veja abaixo) e é muito flexível, permitindo a estimativa com falta observações, previsões, funções de resposta a impulsos e muito mais.

Exemplo: Modelo AR(2)

Um modelo autoregressivo é um bom exemplo introdutório para colocar modelos na forma de espaço de estado. Um modelo AR (2) geralmente é escrito como:

$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma^2)$

Vamos colocar na forma do espaço de estado, para isso note que

$$\begin{bmatrix} y_t \\ y_{t-1} \end{bmatrix} = \begin{bmatrix} \phi_1 & \phi_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y_{t-1} \\ y_{t-2} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \epsilon_t.$$

Defina $\alpha_t = (y_t, y_{t-1})^\top$. Logo, podemos escrever as seguintes equações

Equação de medida:

$$y_t = \begin{bmatrix} 1 & 0 \end{bmatrix} \alpha_t$$

Equação de transição:

$$\alpha_{t+1} = \begin{bmatrix} \phi_1 & \phi_2 \\ 1 & 0 \end{bmatrix} \alpha_t + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \eta_t$$

Na forma geral do modelo de espaço de estado

$$\begin{aligned} y_t &= Z_t \alpha_t + d_t + \varepsilon_t \\ \alpha_{t+1} &= T_t \alpha_t + c_t + R_t \eta_t \end{aligned}$$

temos

$Z_t \equiv Z = \begin{bmatrix} 1 & 0 \end{bmatrix}$, $T_t \equiv T = \begin{bmatrix} \phi_1 & \phi_2 \\ 1 & 0 \end{bmatrix}$, $R_t \equiv R = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\eta_t \equiv \epsilon_{t+1} \sim N(0, \sigma^2)$ e os demais termos nulos.

Os três parâmetros desconhecidos desse modelo são ϕ_1, ϕ_2, σ^2.

1.3 Alguns modelos de espaço de estado específicos

Para maior profundidade, ver Franco et. al (2009) (ftp://www.est.ufmg.br/pub/glaura/Series%20Temporais/M
MEE-ESTE.pdf).

- Modelo de nível local (MNL)
- Modelo de tendência local (MTL)
- Modelo de tendência polinomial (MTP)
- Modelo estrutural básico (MEB): tendência + sazonalidade

1.4 Métodos de estimação para o modelo de espaço de estado

Alguns métodos são utilizados para a estimação de parâmetros e vetor de estados

- Filtro de Kalman: Um algoritmo recursivo que determina a estimativa do vetor de estados no tempo t dada toda a informação disponıvel até o instante $t-1$. https://pt.wikipedia.org/wiki/Filtro_de_Kalman
- Estimação de Máxima verossimilhança na abordagem clássica, em geral numericamente, com um algoritmo de otimização não linear, por exemplo BFGS
- Markov chain Monte Carlo (MCMC) na abordagem Bayesiana, média a posteriori, mediana a posteriori, moda a posteriori, estimador de Bayes.

1.5 Modelos SARIMAX

Fonte: https://www.statsmodels.org/stable/examples/notebooks/generated/statespace_sarimax_stata.html

Motivação

Considere os dados apresentados no artigo

Friedman, M., and D. Meiselman. 1963. The relative stability of monetary velocity and the investment multiplier in the United States, 1897–1958. In Stabilization Policies, Commission on Money and Credit, 123–126. EnglewoodCliffs, NJ: Prentice Hall.

Os autores postulam uma relação direta entre as despesas de consumo pessoal (consump) e o suprimento de dinheiro medido por M2 (m2)

$$consumo = \beta_0 + \beta_1 m2_t + \epsilon_t$$

Como definir um modelo com variáveis exógenas com erros SARIMA?

Uma proposta é considerar

$$y_t = \beta_0 + \beta_1 x_t + u_t$$

$$\phi_p(B)\Phi_P(B^m)\Delta^d\Delta_m^D u_t = A(t) + \theta_q(B)\Theta_Q(B^m)a_t$$

que podemos colocar na forma dos modelos de espaço de estado que falamos acima.

1.5.1 Aplicação

[ ]:
```
import numpy as np
import pandas as pd
from scipy.stats import norm
import statsmodels.api as sm
import matplotlib.pyplot as plt
from datetime import datetime
import requests
from io import BytesIO

# Evitar warnings não prejudiciais
pd.plotting.register_matplotlib_converters()

%matplotlib inline
```

[ ]:
```
# Leitura dos dados
friedman2 = requests.get('https://www.stata-press.com/data/r12/friedman2.dta').
 ↪content
data = pd.read_stata(BytesIO(friedman2))
data.index = data.time
data.index.freq = "QS-OCT"

data
```

[ ]:
```
data.index
```

[ ]:
```
# Visualização dos dados

data['consump'].plot(label='Consumo', legend=True)
```

[ ]:
```
# Explorando a associação entre as variáveis

plt.plot(data['m2'],data['consump'])
```

[ ]:
```
# Definição de variáveis endógenas e exógenas + inclusão de intercepto para exog

endog = data.loc['1959':'1981', 'consump']
exog = sm.add_constant(data.loc['1959':'1981', 'm2'])
```

[ ]:
```
exog.head()
```

[ ]:
```
# Vamos ajustar um modelo ARMA(1,1) considerando os dados de 1959 a 1981
# Veja https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.
 ↪sarimax.SARIMAX.html

modelo = sm.tsa.statespace.SARIMAX(endog, exog, order=(1,0,1))
resultado = modelo.fit(disp=False)
print(resultado.summary())
```

[ ]:
```
# Base de dados até 1981

raw = pd.read_stata(BytesIO(friedman2))
raw.index = raw.time
raw.index.freq = "QS-OCT"
data = raw.loc[:'1981']
```

[ ]:
```
data
```

[ ]:
```
# Variáveis endógenas e exógenas
endog = data.loc['1959':, 'consump']
exog = sm.add_constant(data.loc['1959':, 'm2'])
nobs = endog.shape[0]

# Ajuste do modelo até 1978
modelo = sm.tsa.statespace.SARIMAX(endog.loc[:'1978-01-01'], exog=exog.loc[:
 ↪'1978-01-01'], order=(1,0,1))
fit_resultado = modelo.fit(disp=False)
print(fit_resultado.summary())
```

[ ]:
```
modelo = sm.tsa.statespace.SARIMAX(endog, exog=exog, order=(1,0,1))
resultado = modelo.filter(fit_resultado.params)
```

[ ]:
```
# Previsão um passo a frente com intervalo de previsão

previsao = resultado.get_prediction()
previsao_ip = previsao.conf_int()
```

[ ]:
```
# Previsão dinâmica

data_previsao_dinamica = '1978-01-01'

previsao_di = resultado.get_prediction(dynamic=data_previsao_dinamica)
previsao_di_ip = previsao_di.conf_int()
```

[ ]:
```
# Visualização dos resultados

# Sobre previsão passo à frente e previsão dinâmica:
# Jackson, E. A. (2018). Comparison between static and dynamic forecast in␣
 ↪autoregressive integrated moving average for seasonally adjusted headline␣
 ↪consumer price index. Available at SSRN 3162606. Disponível em https://mpra.ub.
 ↪uni-muenchen.de/86180/1/MPRA_paper_86180.pdf. Acessado em 01/08/2020.


fig, ax = plt.subplots(figsize=(9,4))
npre = 4
ax.set(title='Consumo pessoal', xlabel='Data', ylabel='Bilhões de dólares')

# Plota os dados
data.loc['1977-07-01':, 'consump'].plot(ax=ax, style='o', label='Observado')

# Previsões
previsao.predicted_mean.loc['1977-07-01':].plot(ax=ax, style='r--',␣
 ↪label='Previsão um passo  à frente')
ip = previsao_ip.loc['1977-07-01':]
ax.fill_between(ip.index, ip.iloc[:,0], ip.iloc[:,1], color='r', alpha=0.1)

previsao_di.predicted_mean.loc['1977-07-01':].plot(ax=ax, style='g',␣
 ↪label='Previsão dinâmica  %s' % data_previsao_dinamica )
ip = previsao_di_ip.loc['1977-07-01':]
ax.fill_between(ip.index, ip.iloc[:,0], ip.iloc[:,1], color='g', alpha=0.1)

legend = ax.legend(loc='lower right')
```

[ ]:
```
# Visualização dos resultados

fig, ax = plt.subplots(figsize=(9,4))
npre = 4
ax.set(title='Consumo pessoal', xlabel='Data', ylabel='Bilhões de dólares')
```

```
# Plota os dados
data['consump'].plot(ax=ax, label='Observado')

# Previsões
previsao.predicted_mean.plot(ax=ax, style='r--', label='Previsão um passo à␣
↪frente')
ip = previsao_ip
ax.fill_between(ip.index, ip.iloc[:,0], ip.iloc[:,1], color='r', alpha=0.1)

previsao_di.predicted_mean.plot(ax=ax, style='g', label='Previsão dinâmica %s'␣
↪% data_previsao_dinamica )
ip = previsao_di_ip
ax.fill_between(ip.index, ip.iloc[:,0], ip.iloc[:,1], color='g', alpha=0.1)

legend = ax.legend(loc='lower right')
```

[ ]:
```
# Erro de predição

fig, ax = plt.subplots(figsize=(9,4))
npre = 4
ax.set(title='Erro de previsão', xlabel='Data', ylabel='Previsão')

# Previsão um passo à frente com intervalos de 95% de confiança
erro_previsao = previsao.predicted_mean - endog
erro_previsao.loc['1977-10-01':].plot(ax=ax, label='Previsão um passo à frente')

ip = previsao_ip.loc['1977-10-01':].copy()
ip.iloc[:,0] -= endog.loc['1977-10-01':]
ip.iloc[:,1] -= endog.loc['1977-10-01':]

ax.fill_between(ip.index, ip.iloc[:,0], ip.iloc[:,1], alpha=0.1)

# Previsão dinâmica com intervalos de 95% de confiança
erro_previsao_di = previsao_di.predicted_mean - endog
erro_previsao_di.loc['1977-10-01':].plot(ax=ax, style='r', label='Previsão␣
↪dinâmica %s' % data_previsao_dinamica )

ip_di = previsao_di_ip.loc['1977-10-01':].copy()
ip_di.iloc[:,0] -= endog.loc['1977-10-01':]
ip_di.iloc[:,1] -= endog.loc['1977-10-01':]

ax.fill_between(ip_di.index, ip_di.iloc[:,0], ip_di.iloc[:,1], color='r',␣
↪alpha=0.1)

legend = ax.legend(loc='lower left');
legend.get_frame().set_facecolor('w')
```

Exercício: mude a data da previsão dinâmica e verifique como se alteram as previsões e os erros de previsão

1.6 Previsão Bayesiana

Leituras adicionais recomendada:

- Pacote Tensorflow Probability: https://www.tensorflow.org/probability
- Modelagem de séries temporais estruturais: https://blog.tensorflow.org/2019/03/structural-time-series-modeling-in.html
- Harvey, A. C. (1989). Forecasting, structural time series models and the Kalman filter. Cambridge University Press.

Os modelos de séries temporais estruturais (STS) são vistos como uma família de modelos de probabilidade que incluem

- processos autorregressivos,
- médias móveis,
- tendências lineares locais,
- sazonalidade e
- regressão e seleção de variáveis em covariáveis externas (outras séries temporais potencialmente relacionadas às séries de interesse).

De uma forma bem simplificada, um modelo de séries temporais estruturais considera que a série temporal pode ser escrita como uma soma de n componentes

$$f(t) = f_1(t) + f_2(t) + \ldots + f_n(t) + \epsilon_t, \;\; \epsilon_t \sim N(0, \sigma^2)$$

A modelagem bayesiana assume que os parâmetros desse modelo seguem distribuições de probabilidade (distribuições a priori) e, a partir da distribuição conjunta dos parâmetros e dados, obtém estimativas a partir da distribuição a posteriori dos parâmetros dadas as observações. No pacote tensorflow_probability são implementados métodos de inferência variacional e Monte Carlo Hamiltoniano.

1.6.1 Aplicação

Fonte da implementação:

https://github.com/tensorflow/probability/blob/master/tensorflow_probability/examples/jupyter_notebook

Fonte original dos dados de CO2:

https://scrippsco2.ucsd.edu/data/atmospheric_co2/primary_mlo_co2_record.html

C. D. Keeling, S. C. Piper, R. B. Bacastow, M. Wahlen, T. P. Whorf, M. Heimann, and H. A. Meijer, Atmospheric CO2 and 13CO2 exchange with the terrestrial biosphere and oceans from 1978 to 2000: observations and carbon cycle implications, pages 83-113, in "A History of Atmospheric CO2 and its effects on Plants, Animals, and Ecosystems", editors, Ehleringer, J.R., T. E. Cerling, M. D. Dearing, Springer Verlag, New York, 2005.

[1]:
```
import pandas as pd
import numpy as np

%matplotlib inline
import matplotlib as mpl
from matplotlib import pylab as plt
import matplotlib.dates as mdates
import seaborn as sns

import collections

import numpy as np
import tensorflow.compat.v2 as tf
import tensorflow_probability as tfp

from tensorflow_probability import distributions as tfd
from tensorflow_probability import sts


# Evitar warnings não prejudiciais
pd.plotting.register_matplotlib_converters()
```

[2]:
```
# Algumas funções auxiliares e definição de estilo para a visualização dos dados␣
 ↪e de previsões

# Fonte: https://github.com/tensorflow/probability/blob/master/
 ↪tensorflow_probability/examples/jupyter_notebooks/
 ↪Structural_Time_Series_Modeling_Case_Studies_Atmospheric_CO2_and_Electricity_Demand.
 ↪ipynb

from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()

sns.set_context("notebook", font_scale=1.)
sns.set_style("whitegrid")
%config InlineBackend.figure_format = 'retina'


def plot_forecast(x, y,
                  forecast_mean, forecast_scale, forecast_samples,
                  title, x_locator=None, x_formatter=None):
  """Plot a forecast distribution against the 'true' time series."""
  colors = sns.color_palette()
  c1, c2 = colors[0], colors[1]
  fig = plt.figure(figsize=(12, 6))
  ax = fig.add_subplot(1, 1, 1)
```

```
  num_steps = len(y)
  num_steps_forecast = forecast_mean.shape[-1]
  num_steps_train = num_steps - num_steps_forecast


  ax.plot(x, y, lw=2, color=c1, label='ground truth')

  forecast_steps = np.arange(
      x[num_steps_train],
      x[num_steps_train]+num_steps_forecast,
      dtype=x.dtype)

  ax.plot(forecast_steps, forecast_samples.T, lw=1, color=c2, alpha=0.1)

  ax.plot(forecast_steps, forecast_mean, lw=2, ls='--', color=c2,
           label='forecast')
  ax.fill_between(forecast_steps,
                  forecast_mean-2*forecast_scale,
                  forecast_mean+2*forecast_scale, color=c2, alpha=0.2)

  ymin, ymax = min(np.min(forecast_samples), np.min(y)), max(np.
 ↪max(forecast_samples), np.max(y))
  yrange = ymax-ymin
  ax.set_ylim([ymin - yrange*0.1, ymax + yrange*0.1])
  ax.set_title("{}".format(title))
  ax.legend()

  if x_locator is not None:
    ax.xaxis.set_major_locator(x_locator)
    ax.xaxis.set_major_formatter(x_formatter)
    fig.autofmt_xdate()

  return fig, ax


def plot_components(dates,
                    component_means_dict,
                    component_stddevs_dict,
                    x_locator=None,
                    x_formatter=None):
  """Plot the contributions of posterior components in a single figure."""
  colors = sns.color_palette()
  c1, c2 = colors[0], colors[1]

  axes_dict = collections.OrderedDict()
  num_components = len(component_means_dict)
  fig = plt.figure(figsize=(12, 2.5 * num_components))
```

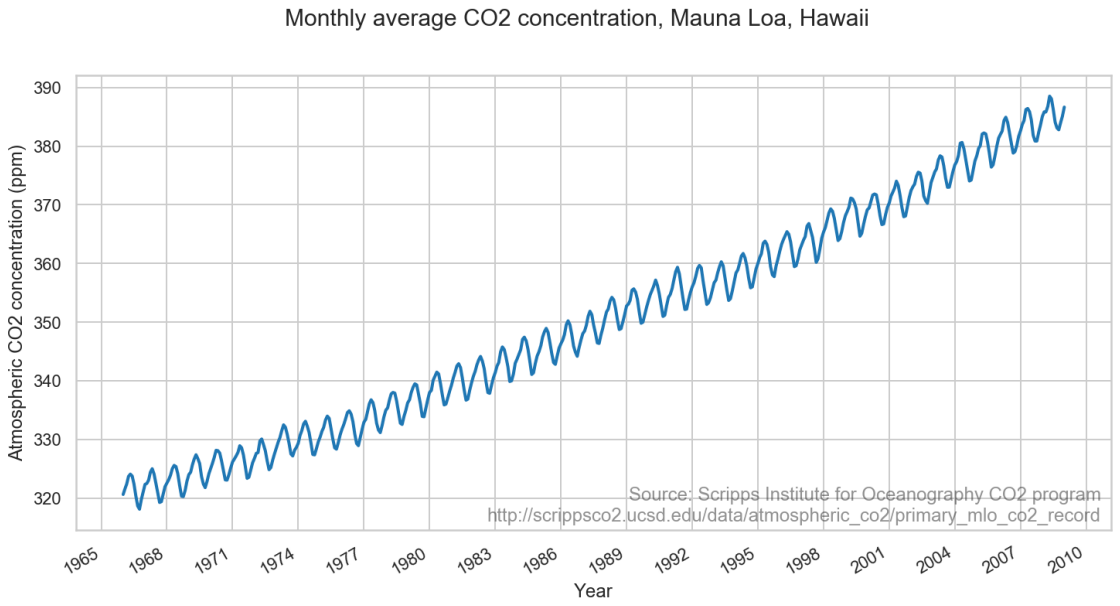
```
  for i, component_name in enumerate(component_means_dict.keys()):
    component_mean = component_means_dict[component_name]
    component_stddev = component_stddevs_dict[component_name]

    ax = fig.add_subplot(num_components,1,1+i)
    ax.plot(dates, component_mean, lw=2)
    ax.fill_between(dates,
                    component_mean-2*component_stddev,
                    component_mean+2*component_stddev,
                    color=c2, alpha=0.5)
    ax.set_title(component_name)
    if x_locator is not None:
      ax.xaxis.set_major_locator(x_locator)
      ax.xaxis.set_major_formatter(x_formatter)
    axes_dict[component_name] = ax
  fig.autofmt_xdate()
  fig.tight_layout()
  return fig, axes_dict
```

[3]:
```
# Leitura dos dados e estabelecimento do índice

# CO2 readings from Mauna Loa observatory, monthly beginning January 1966
# Original source: http://scrippsco2.ucsd.edu/data/atmospheric_co2/
 ↪primary_mlo_co2_record
```

```
co2_by_month = np.array('320.62,321.60,322.39,323.70,324.08,323.75,322.38,320.
 ↪36,318.64,318.10,319.78,321.03,322.33,322.50,323.04,324.42,325.00,324.09,322.
 ↪54,320.92,319.25,319.39,320.73,321.96,322.57,323.15,323.89,325.02,325.57,325.
 ↪36,324.14,322.11,320.33,320.25,321.32,322.89,324.00,324.42,325.63,326.66,327.
 ↪38,326.71,325.88,323.66,322.38,321.78,322.85,324.12,325.06,325.98,326.93,328.
 ↪14,328.08,327.67,326.34,324.69,323.10,323.06,324.01,325.13,326.17,326.68,327.
 ↪17,327.79,328.92,328.57,327.36,325.43,323.36,323.56,324.80,326.01,326.77,327.
 ↪63,327.75,329.73,330.07,329.09,328.04,326.32,324.84,325.20,326.50,327.55,328.
 ↪55,329.56,330.30,331.50,332.48,332.07,330.87,329.31,327.51,327.18,328.16,328.
 ↪64,329.35,330.71,331.48,332.65,333.09,332.25,331.18,329.39,327.43,327.37,328.
 ↪46,329.57,330.40,331.40,332.04,333.31,333.97,333.60,331.90,330.06,328.56,328.
 ↪34,329.49,330.76,331.75,332.56,333.50,334.58,334.88,334.33,333.05,330.94,329.
 ↪30,328.94,330.31,331.68,332.93,333.42,334.70,336.07,336.75,336.27,334.92,332.
 ↪75,331.59,331.16,332.40,333.85,334.97,335.38,336.64,337.76,338.01,337.89,336.
 ↪54,334.68,332.76,332.55,333.92,334.95,336.23,336.76,337.96,338.88,339.47,339.
 ↪29,337.73,336.09,333.92,333.86,335.29,336.73,338.01,338.36,340.07,340.77,341.
 ↪47,341.17,339.56,337.60,335.88,336.02,337.10,338.21,339.24,340.48,341.38,342.
 ↪51,342.91,342.25,340.49,338.43,336.69,336.86,338.36,339.61,340.75,341.61,342.
 ↪70,343.57,344.14,343.35,342.06,339.81,337.98,337.86,339.26,340.49,341.38,342.
 ↪52,343.10,344.94,345.76,345.32,343.98,342.38,339.87,339.99,341.15,342.99,343.
 ↪70,344.50,345.28,347.06,347.43,346.80,345.39,343.28,341.07,341.35,342.98,344.
 ↪22,344.97,345.99,347.42,348.35,348.93,348.25,346.56,344.67,343.09,342.80,344.
 ↪24,345.56,346.30,346.95,347.85,349.55,350.21,349.55,347.94,345.90,344.85,344.
 ↪17,345.66,346.90,348.02,348.48,349.42,350.99,351.85,351.26,349.51,348.10,346.
 ↪45,346.36,347.81,348.96,350.43,351.73,352.22,353.59,354.22,353.79,352.38,350.
 ↪43,348.73,348.88,350.07,351.34,352.76,353.07,353.68,355.42,355.67,355.12,353.
 ↪90,351.67,349.80,349.99,351.30,352.52,353.66,354.70,355.38,356.20,357.16,356.
 ↪23,354.81,352.91,350.96,351.18,352.83,354.21,354.72,355.75,357.16,358.60,359.
 ↪34,358.24,356.17,354.02,352.15,352.21,353.75,354.99,355.99,356.72,357.81,359.
 ↪15,359.66,359.25,357.02,355.00,353.01,353.31,354.16,355.40,356.70,357.17,358.
 ↪38,359.46,360.28,359.60,357.57,355.52,353.69,353.99,355.34,356.80,358.37,358.
 ↪91,359.97,361.26,361.69,360.94,359.55,357.48,355.84,356.00,357.58,359.04,359.
 ↪97,361.00,361.64,363.45,363.80,363.26,361.89,359.45,358.05,357.75,359.56,360.
 ↪70,362.05,363.24,364.02,364.71,365.41,364.97,363.65,361.48,359.45,359.61,360.
 ↪76,362.33,363.18,363.99,364.56,366.36,366.80,365.63,364.47,362.50,360.19,360.
 ↪78,362.43,364.28,365.33,366.15,367.31,368.61,369.30,368.88,367.64,365.78,363.
 ↪90,364.23,365.46,366.97,368.15,368.87,369.59,371.14,371.00,370.35,369.27,366.
 ↪93,364.64,365.13,366.68,368.00,369.14,369.46,370.51,371.66,371.83,371.69,370.
 ↪12,368.12,366.62,366.73,368.29,369.53,370.28,371.50,372.12,372.86,374.02,373.
 ↪31,371.62,369.55,367.96,368.09,369.68,371.24,372.44,373.08,373.52,374.85,375.
 ↪55,375.40,374.02,371.48,370.70,370.25,372.08,373.78,374.68,375.62,376.11,377.
 ↪65,378.35,378.13,376.61,374.48,372.98,373.00,374.35,375.69,376.79,377.36,378.
```

```
co2_by_month = co2_by_month
num_forecast_steps = 12 * 10 # Vamos predizer os dados dos últimos 10 anos
co2_by_month_training_data = co2_by_month[:-num_forecast_steps]

co2_dates = np.arange("1966-01", "2019-02", dtype="datetime64[M]")
co2_loc = mdates.YearLocator(3)
co2_fmt = mdates.DateFormatter('%Y')
```

[4]:
```
# Visualização dos dados

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.plot(co2_dates[:-num_forecast_steps], co2_by_month_training_data, lw=2,␣
 ↪label="training data")
ax.xaxis.set_major_locator(co2_loc)
ax.xaxis.set_major_formatter(co2_fmt)
ax.set_ylabel("Atmospheric CO2 concentration (ppm)")
ax.set_xlabel("Year")
fig.suptitle("Monthly average CO2 concentration, Mauna Loa, Hawaii",
             fontsize=15)
ax.text(0.99, .02,
        "Source: Scripps Institute for Oceanography CO2 program\nhttp://
 ↪scrippsco2.ucsd.edu/data/atmospheric_co2/primary_mlo_co2_record",
        transform=ax.transAxes,
        horizontalalignment="right",
        alpha=0.5)
fig.autofmt_xdate()
```

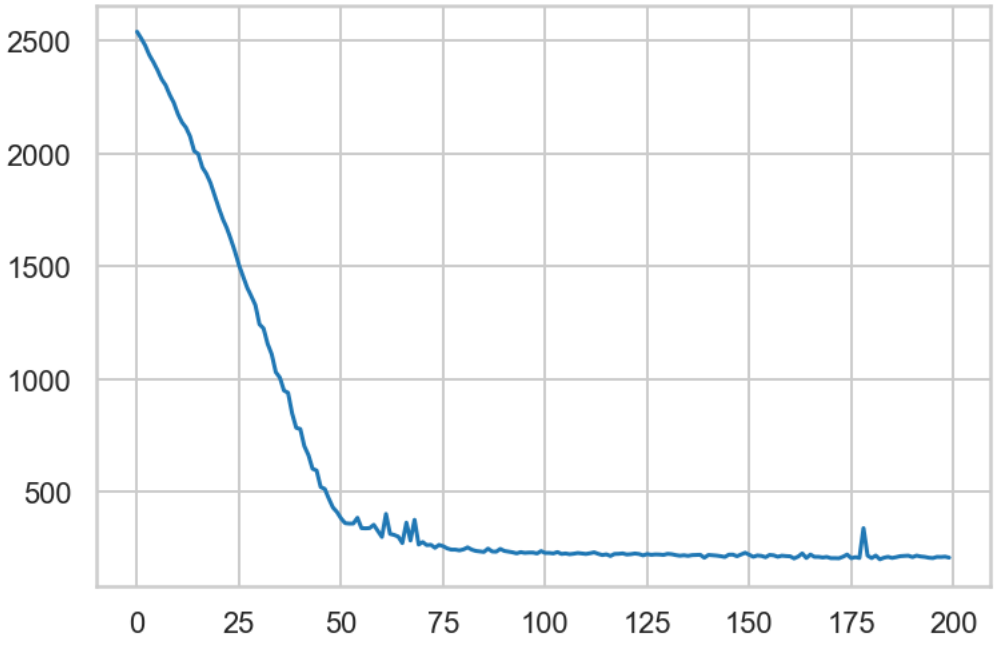


[5]:
```
# função para ajuste de um modelo com tendência e sazonalidade
# Leituras adicionais:
# https://www.tensorflow.org/probability/api_docs/python/tfp/sts/LocalLinearTrend
# https://www.tensorflow.org/probability/api_docs/python/tfp/sts/Seasonal


def build_model(observed_time_series):
  trend = sts.LocalLinearTrend(observed_time_series=observed_time_series)
  seasonal = tfp.sts.Seasonal(
      num_seasons=12, observed_time_series=observed_time_series)
  model = sts.Sum([trend, seasonal], observed_time_series=observed_time_series)
  return model
```

[9]:
```
# Ajuste do modelo usando o pacote tensorflow_probability

co2_model = build_model(co2_by_month_training_data)

# Build the variational surrogate posteriors `qs`.
variational_posteriors = tfp.sts.build_factored_surrogate_posterior(
    model=co2_model)
```

[10]:
```
#@title Minimize the variational loss.
# Referências para ELBO e método Bayesiano Variacional:
# https://en.wikipedia.org/wiki/Evidence_lower_bound
# https://en.wikipedia.org/wiki/Variational_Bayesian_methods

# Allow external control of optimization to reduce test runtimes.
num_variational_steps = 200 # @param { isTemplate: true}
num_variational_steps = int(num_variational_steps)

optimizer = tf.optimizers.Adam(learning_rate=.1)
# Using fit_surrogate_posterior to build and optimize the variational loss␣
 ↪function.
@tf.function(experimental_compile=True)
def train():
  elbo_loss_curve = tfp.vi.fit_surrogate_posterior(
    target_log_prob_fn=co2_model.joint_log_prob(
        observed_time_series=co2_by_month_training_data),
    surrogate_posterior=variational_posteriors,
    optimizer=optimizer,
    num_steps=num_variational_steps)
  return elbo_loss_curve

elbo_loss_curve = train()
```

```
plt.plot(elbo_loss_curve)
plt.show()

# Draw samples from the variational posterior.
q_samples_co2_ = variational_posteriors.sample(50)
```



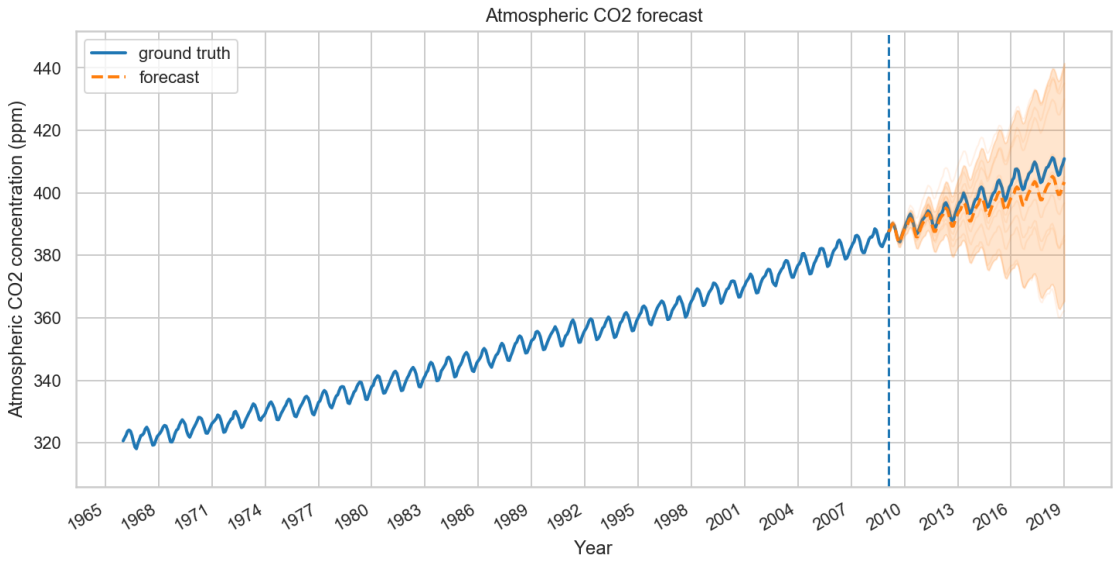
[11]:
```
print("Inferred parameters:")
for param in co2_model.parameters:
  print("{}: {} +- {}".format(param.name,
                              np.mean(q_samples_co2_[param.name], axis=0),
                              np.std(q_samples_co2_[param.name], axis=0)))
```

```
Inferred parameters:
observation_noise_scale: 0.19643934071063995 +- 0.006863068789243698
LocalLinearTrend/_level_scale: 0.11734524369239807 +- 0.026956036686897278
LocalLinearTrend/_slope_scale: 0.02262195385992527 +- 0.0053810253739356995
Seasonal/_drift_scale: 0.04438478872179985 +- 0.007635717745870352
```

[12]:
```
# Previsão

co2_forecast_dist = tfp.sts.forecast(
    co2_model,
    observed_time_series=co2_by_month_training_data,
    parameter_samples=q_samples_co2_,
    num_steps_forecast=num_forecast_steps)
```

[13]:
```
num_samples=10

co2_forecast_mean, co2_forecast_scale, co2_forecast_samples = (
    co2_forecast_dist.mean().numpy()[..., 0],
    co2_forecast_dist.stddev().numpy()[..., 0],
    co2_forecast_dist.sample(num_samples).numpy()[..., 0])
```

[14]:
```
fig, ax = plot_forecast(
    co2_dates, co2_by_month,
    co2_forecast_mean, co2_forecast_scale, co2_forecast_samples,
    x_locator=co2_loc,
    x_formatter=co2_fmt,
    title="Atmospheric CO2 forecast")
ax.axvline(co2_dates[-num_forecast_steps], linestyle="--")
ax.legend(loc="upper left")
ax.set_ylabel("Atmospheric CO2 concentration (ppm)")
ax.set_xlabel("Year")
fig.autofmt_xdate()
```



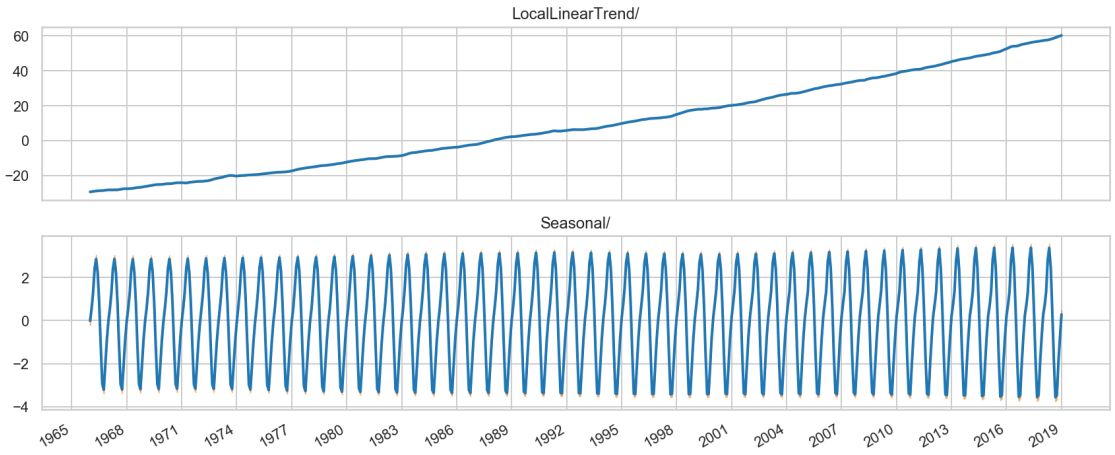
[15]:
```
# Build a dict mapping components to distributions over
# their contribution to the observed signal.
component_dists = sts.decompose_by_component(
    co2_model,
    observed_time_series=co2_by_month,
    parameter_samples=q_samples_co2_)
```

[16]:
```
co2_component_means_, co2_component_stddevs_ = (
    {k.name: c.mean() for k, c in component_dists.items()},
```

```
    {k.name: c.stddev() for k, c in component_dists.items()})
```

[17]:
```
plot_components(co2_dates, co2_component_means_, co2_component_stddevs_,
                x_locator=co2_loc, x_formatter=co2_fmt)
```

[17]:
```
(<Figure size 864x360 with 2 Axes>,
 OrderedDict([('LocalLinearTrend/',
               <matplotlib.axes._subplots.AxesSubplot at 0x7f6ba861b090>),
              ('Seasonal/',
               <matplotlib.axes._subplots.AxesSubplot at 0x7f6ba8101090>)]))
```



[ ]:
```

```