**Introduction**

Curve fitting can capture trends in the data allowing to make predictions based on this trend. Splines are often employed for smoothing and fitting more complex curves. Splines are cubic polynomial functions that fit between two neighboring cut points. However, the cut points must be set a priori. The placement of the cut point, or knot, affects what data are used for spline curve fitting. Bayesian Adaptive Regression Splines (BARS) was developed to adaptively determine knot sets within regression splines via Markov Chain Monte Carlo (MCMC) simulations [1]. BARS was originally developed in 2001 by DiMatteo et al [2]. Later it was implemented in C with R and S wrappers in 2008 [1]. It was much faster code than the original in S. It could be generally used in Normal or Poisson regression models. It was also adapted to fit hierarchical Bayesian models in 2007 [3]. It has been used in a variety of applications such as neurophysiology, imaging, genetics, and EEG analysis [4, 5, 6]. The BARS algorithm can adapt knot positions by adding, removing, or relocating them to more accurately estimate a given curve. I aim to develop python code that mimics the Wallstrom, Liebner, and Kass paper’s [1] algorithm and apply its example dataset.

**Methods**

For BARS we can take the data and assign a Hierarchical Bayesian probability model

Where is our spline basis function and represents the data.

On some time interval [A, B] we have data, where each observation has data over where each is assumed to depend probabilistically on . is the random variable for the observed patient data. These function evaluations can be collected into a vector where is the design matrix of the spline basis functions evaluated at the data and the coefficient vector representing the spline basis coefficient given a knot set .

In Bayesian parameter inference we want the posterior distribution of the model parameters, which are representing the coefficients of the spline basis, the spline set, the number of knots, k, and the nuisance parameter.

Where

This is the marginal density of the posterior whereis marginalized out. RJMCMC performs sampling on for this integral.

RJMCMC sampling is done from the marginal posterior on to get our knot sets. First we define the prior distributions

Where . We use uniform priors onTo sample these, integrate out of the posterior. Given these priors we can analytically compute the integral above or the marginal posterior. From this we can form likelihood ratio for candidate knots

For each draw of knots from the posterior, a draw is obtained from the conditional posterior of conditionally on We use this draw to determine the value of to get our vector. The conditional posterior of can be assumed normal and the initial sample is taken from the normal distribution. The cubic basis functions are built with the given knot draw and the design matrix, , is extracted by evaluating it at the data points. Once a fit is acquired, BIC is found. If the BIC for the candidate model is larger than the prior model on a different knot set. Then the candidate model is set as the current one. This is repeated for some number of MCMC iterations.

On Initialization you input the necessary parameters such as the maximum number of possible knots, and the number of starting knots desired. The data is then inputted and normalized. An initial knot set is computed.

Then a base model is fit using the methods described above. Given this information we can calculate the birth and death probabilities given each number of knots k by

to determine if there is a birth, death, or relocation of a specific knot. The probability is sampled from uniform(0,k). c is a defined parameter used to control these probabilities. Once you have all the knot probabilities up to some maximum number, if there is a birth step it will then draw an existing knot from the set and center it around a beta distribution to draw a random sample from to be the new candidate knot. Then given a knot set on an iteration it will then fit a candidate model. After this generation of the random beta vector is done. This is the posterior distribution. It generates from a multivariate normal distribution.

**Algorithm:** Bayesian Adaptive Regression Splines

Initialize parameters

Read data and normalize

Initialize knots

Initialize Model

Calculate birth and death probabilities for each possible value of *k*

**if** k==maxknots **then**

birth probability = 0

**else** birth probability = c \* min(1, p(k+1)/p(k))

**end if**

**if** k==1 **then**

death probability = 0

**else** death probability = c \* min(1, p(k-1)/p(k))

**end if**

Run RJMCMC

**for** *i* = 1*,...,iter* **do**

Take knot step based on probability: addition, deletion, or relocation. Produces *ξ*

u ∼ U(0,1)

**if** u *<* birth probability **then**

s ∼ Discrete Uniform(*ξcurr*) t ∼ beta(*sτ,*(1 − *s*)*τ*)

*ξcand* = *ξcurr* ∪ *t*

k = k + 1

Obtain marginal density posteri

Form spline basis and fit the model given *ξcand*

Obtain marginal density posterior to determine if the new model is selected

**else if** 1 - u *<* death probability **then**

s ∼ Discrete Uniform(*ξcurr*)

*ξcand* = *ξcurr* \ *s*

k = k - 1

Form spline basis and fit the model given *ξcand*

Obtain marginal density posterior to determine if the new model is selected

**else**

s ∼ Discrete Uniform(*ξcurr*) t ∼ beta(*sτ,*(1 − *s*)*τ*)

*ξcand* = (*ξcurr* ∪ *t*) \ *s* k = k

Form spline basis and fit the model given *ξcand*

Obtain marginal density posterior to determine if the new model is selected

**end if**

**end for**

Obtain final model for further analyses

Using this algorithm we can find an optimal set of knots for a Normal model. Then we compute the random vector of the betas based on the knot set found.

**Data Application**

The data come from a neuron in inferotemporal cortex recorded using 16 trials of an experiment [7]. Figure 1 shows a dataset over time and the initial knot set used.

A graph with blue lines

AI-generated content may be incorrect. A graph with blue lines

AI-generated content may be incorrect.

**Figure 1. Normal Dataset.** Shows the dataset (t, y(t)) that we want to fit a spline to. Second figure shows the data with 25 initial knots evenly spaces from 0 to 1

Once the base algorithm is implemented computational issues arise specifically singularity issues. Samples knots will grow closer to each other until the design matrix is not positive definite and it fails. The first plot in Figure 2 shows this before it happens. This can be fixed be adding a knot checking process if a knot is too close to another knot resample it. Similarly the condition number of the matrix can come out greater than 1 which indicates issues with solving the normal model exactly. The second plot in Figure 2 shows the sparsity of the knot placement and it focusing more on one end overfitting it. This issue could not be resolved.

A graph with lines and a red line

AI-generated content may be incorrect. A graph with a red line

AI-generated content may be incorrect.

**Figure 2. Computational Issues.** Shows the errors in computation of MCMC algorithm.

The resulting Figure 3 would occur some of the time where it appears it only adapted the knot placement a few times and did not operate correctly.

A graph with lines and a red line

AI-generated content may be incorrect.

**Figure 3. Result.** Shows the result of the algorithm.

**Conclusion**

The algorithm of BARS is very powerful and useful for finding optimal knot placement given your dataset. Many applications that use splines have the difficulty of finding the a priori placement of knots that are important for the fit and inference. The RJMCMC is used on the knot set after integration of the marginal density. The integration can be performed exactly because we are using a normal model. It allows for the repositioning of the knots because of the birth, relocate, and death steps. A candidate is accepted once sampled if the acceptance probability of the likelihood of the candidate minus the current model fit to the knot set is large enough. To address the computational issues, a checking procedure must be implemented to avoid singularities, and a flag to avoid computations with singular matrices. Although this attempt did not fully work many of the main components were completed and only the exceptions of computational issues were left out. With just the initial knot set the fit is good. Overall this method is a useful adjustment of the RJMCMC to put to the application of optimally adapting knot sets for model fitting.

**References**

1. Wallstrom, G. ., J. . Liebner, and R. E. . Kass. “An Implementation of Bayesian Adaptive Regression Splines (BARS) in C With S and R Wrappers”. Journal of Statistical Software, vol. 26, no. 1, June 2008, pp. 1-21, doi:10.18637/jss.v026.i01.
2. DiMatteo, Ilaria, Christopher R. Genovese, and Robert E. Kass. "Bayesian curve‐fitting with free‐knot splines." Biometrika 88.4 (2001): 1055-1071.
3. Bigelow, Jamie L., and David B. Dunson. "Bayesian adaptive regression splines for hierarchical data." Biometrics 63.3 (2007): 724-732.
4. Kass, Robert E, et al. “Statistical Smoothing of Neuronal Data.” Network: Computation in Neural Systems, vol. 14, no. 1, 2003, pp. 5–15., <https://doi.org/10.1088/0954-898x/14/1/301>.
5. Wallstrom, Garrick L et al. “Automatic correction of ocular artifacts in the EEG: a comparison of regression-based and component-based methods.” International journal of psychophysiology : official journal of the International Organization of Psychophysiology vol. 53,2 (2004): 105-19. doi:10.1016/j.ijpsycho.2004.03.007
6. Zhang, Xiaohua et al. “Integration of association statistics over genomic regions using Bayesian adaptive regression splines.” Human genomics vol. 1,1 (2003): 20-9. doi:10.1186/1479-7364-1-1-20
7. Baker CI, Behrmann M, Olson CR. Impact of Learning on Representation of Parts and Wholes in Monkey Inferotemporal Cortex. Nature Neuroscience 2002;5(11):1210–1216.

**Appendix: Python Code**

# made with python version 3.9.16

# pip install pandas

# pip install matplotlib

# pip install scipy

# pip install BSpline

# pip install openpyxl

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from scipy.interpolate import BSpline

from scipy import stats

from scipy import linalg

#os.chdir('C://Users//mzgra//OneDrive//Dissertation Research//Mike Grabel//Data and Code')

os.chdir(r'C:\Users\mzgra\OneDrive\Dissertation Research\BARS Python')

data = pd.read\_excel('NormalData.xlsx', header = None)

x = data[0] # main variable to perform splines on

y = data[1]

#plt.plot(x, y)

n = len(x)

# normalize x

def normalize(x):

nx = (x - min(x))/(max(x)-min(x))

return nx

nx = normalize(x)

# parameers

# initialize knots

def initialknots(k):

knots = np.zeros(k)

for i in range(k):

knots[i] = (i+1)/(k+1)

return knots

#plt.plot(nx,y)

# for i in knots:

# plt.axvline(x=i, ls = '--')

# plt.show()

def SetRJprobs(c, maxknots):

# Set probabilities of adding or removing a knot given the number of knots

birthprob = np.zeros(maxknots)

deathprob = np.zeros(maxknots)

for k in range(maxknots):

if k == (maxknots-1): # if number of knots = maxknots prob of adding a new knot is 0

birthprob[k] = 0

else:

a = np.random.uniform(0,(k+1)+1) # calculate the ratio P(k+1)/P(k) by sampling from uniform prior for the probability of generating a new knot at this number of knots

b = np.random.uniform(0,(k+1))

birthprob[k] = c \* min(1, a/b) # multiply by parameter c which controls the probability

if k == 0: # if number of knots = 1 prob of removing a knot is 0

deathprob[k] = 0

else:

b = np.random.uniform(0,(k+1)) # calculate the ratio P(k-1)/P(k)

d = np.random.uniform(0,(k+1)-1)

deathprob[k] = c \* min(1, d/b)

return birthprob, deathprob

def fitnormalmodel(x, y, knots\_ext):

flag = False

X = BSpline.design\_matrix(x, knots\_ext, 3)#, extrapolate=True) # fit spline basis at knot points

X = X[:,2:]

aX = X.toarray()

aXt = np.transpose(aX) # get X'

Xty = np.dot(aXt, y) # X'Y

XtX = np.dot(aXt, aX) # X'X

cond\_num = np.linalg.cond(XtX)

if cond\_num > 1 / np.finfo(float).eps:

flag = True

return 0,0,0,0,flag

B = linalg.solve(XtX, Xty) # solve (X'X)B = X'Y for B

mu = np.dot(aX, B) # fit model XB = Y

LL = np.sum(stats.norm.logpdf(y, mu)) # get log likelihood of the fit of this model

return aX, B, mu, LL, flag

def checkcand(knot\_cand, knots\_ext):

norm = np.zeros(len(knots\_ext))

for i in range(len(knots\_ext)):

norm[i] = (np.abs(knot\_cand - knots\_ext[i])< 2e-4)

if np.any(norm == 1):

return 1

elif np.all(norm == 0):

return 0

def getdensity(knot\_cand, knots, tau):

d = np.zeros(len(knots))

for i in range(len(knots)):

d[i] = stats.beta.pdf(knot\_cand, tau\*knots[i], (1 - knots[i])\* tau)

D = sum(d)

return D

def getdensities(knot\_cand, s, tau):

dens1 = stats.beta.pdf(knot\_cand, tau\*s, tau\*(1-s))

dens2 = stats.beta.pdf(s, tau\*s, tau\*(1-knot\_cand))

return dens1, dens2

def BARS(nx, y):

tau = 50

c = 0.4

maxknots = 60

k = 25

num\_iter = 25000

[birthprob, deathprob] = SetRJprobs(c, maxknots)

knots = initialknots(k)

# add 0 and 1 as boundary knots

knots\_ext = np.r\_[0,0,0,0, knots, 1,1,1,1]

[aX, B, mu, LL\_curr, flag] = fitnormalmodel(nx, y, knots\_ext)

# for i in range(len(knots)):

# knotsn = knots[knots != knots[i]]

# knotsn\_ext = np.r\_[0,0,0,0, knotsn, 1,1,1,1]

# [aX, B, mu, LL\_cand, flag] = fitnormalmodel(x, y, knotsn\_ext)

# best\_i = -1

# best\_ll = -np.inf

# if best\_i < 0 or LL\_cand > best\_ll:

# best\_ll = LL\_cand

# best\_i = i

# i = best\_i if best\_i >= 0 else 0

# cand = knots[i]

# D = getdensity(cand, knots, tau) # get densities centered at the candidate knot

# acceptance\_prob = np.exp(best\_ll - LL\_curr + np.log(k-1) - np.log(D) - 0.5 \* np.log(n)) # acceptance probability of this model

for i in range(num\_iter):

print(i)

u = np.random.uniform(0,1) # sample from uniform(0,1)

if u < birthprob[k]: # if that sample is less than the birth probability at the current number of knots

cc = 1

while cc == 1:

# birth step build model

s = np.random.choice(knots[knots != 0]) # randomly select a current knot

# center s around a beta dist

alpha = s\*tau #alpha and beta are multiplied by parameter tau which controls the spread for the candidate knots

beta = (1-s)\*tau

t = np.random.beta(alpha, beta, 1000) # generate a beta distrbution centered around the sampled knot

knot\_cand = np.random.choice(t) # randomly select a candidate knot from the distribution

cc = checkcand(knot\_cand, knots\_ext)

xi\_cand = np.sort(np.r\_[knots, knot\_cand]) # append it to the current set of knots

k\_cand = k + 1 # add 1 to the number of knots

# create model with candidate knots

xi\_ext = np.r\_[0,0,0,0, xi\_cand, 1,1,1,1]

[aX, B, mu, LL\_cand, flag] = fitnormalmodel(nx, y, xi\_ext) # fit a normal model given the candidate set of knots

if flag == True:

i -= 1

continue

else:

D = getdensity(knot\_cand, knots, tau) # get densities centered at the candidate knot

acceptance\_prob = np.exp(LL\_cand - LL\_curr + np.log(k) - np.log(D) - 0.5 \* np.log(n)) # acceptance probability of this model

elif (1.0-u) < deathprob[k]: # if 1 - the random sample from uniform is less than the death probability at the current number of knots

# death step build model

s = np.random.choice(knots[knots != 0]) # randomly select a current knot

xi\_cand = np.sort(knots[knots != s]) # remove it from the set of knots

k\_cand = k - 1 # subtract 1 from the number of knots

xi\_ext = np.r\_[0,0,0,0, xi\_cand, 1,1,1,1]

[aX, B, mu, LL\_cand, flag] = fitnormalmodel(nx, y, xi\_ext) # fit a normal model given the candidate set of knots

if flag == True:

i -= 1

continue

else:

D = getdensity(s, xi\_cand, tau) # get densities centered at the candidate knot

acceptance\_prob = np.exp(LL\_cand - LL\_curr + np.log(k\_cand) - np.log(D) - 0.5 \* np.log(n)) # acceptance probaility of this model

else: # else 1 - birth - death relocate a knot

#relocate step

cc = 1

while cc == 1:

s = np.random.choice(knots[knots != 0]) # randomly select a current knot

# center s around a beta dist

alpha = s\*tau

beta = (1-s)\*tau

t = np.random.beta(alpha, beta, 1000)

knot\_cand = np.random.choice(t) # randomly select from the beta distrbution centered at the sample knot

cc = checkcand(knot\_cand, knots\_ext)

xim = knots[knots!=s] # remove this knot

xi\_cand = np.sort(np.r\_[xim, knot\_cand]) # and add the new knot

k\_cand = k # number of knots remains the same

xi\_ext = np.r\_[0,0,0,0, xi\_cand, 1,1,1,1]

[aX, B, mu, LL\_cand, flag] = fitnormalmodel(nx, y, xi\_ext) # fit a normal model given the candidate set of knots

if flag == True:

i -= 1

continue

else:

D1, D2 = getdensities(knot\_cand, s, tau) # get densities

acceptance\_prob = np.exp(LL\_cand - LL\_curr + np.log(D1) - np.log(D2)) # get acceptance probability

u = np.random.uniform(0,1) # randomly select from the uniform(0,1)

if u < acceptance\_prob: # if it is less than the acceptance probability for a given model above switch out the models

# temp = curr

# curr = cand

# cand = temp

LL\_curr = LL\_cand

knots = xi\_cand

k = k\_cand

m = mu # return the fit XB

# return the parameter

return m, aX, B, knots, k, LL\_curr

m, X, B, knots, k, LL\_curr = BARS(nx, y) # returns XB

plt.plot(nx, y, color = 'black')

plt.plot(nx, m, color = 'red')

plt.vlines(x = knots, ymin = -2, ymax = 2, color = 'gray')

def unitprior(B, J, n, p):

log\_prior = 0.0

for i in range(1, p):

log\_prior += -0.5 \* (B[i]\*\*2) / n

return log\_prior

def RandBeta(X, y, B, LL\_curr, knots):

MHT = -10

iters = 0

B\_curr = B

p = np.shape(X)[1]

n = np.shape(X)[0]

for i in range(3):

iters += 1

Xt = np.transpose(X)

XtX = np.dot(Xt, X)

J = linalg.cholesky(XtX)

z = np.random.multivariate\_normal(mean = np.zeros(p), cov = np.identity(p)) # ~ N(0,I)

A = linalg.solve(J, z) # solve J'A = z for A

B\_cand = B + A # new cand beta

mu\_cand = np.dot(X, B\_cand) # fit model XB = Y

# log likelihood

LL\_cand = np.sum(stats.norm.logpdf(y, mu\_cand)) # get log likelihood of the fit of this model

# priors

curgi = unitprior(B\_curr, J, n, p)

lastgi = unitprior(B\_cand, J, n, p)

# normal proposal density priors

XtWX = np.dot(np.dot(Xt, np.diag(mu\_cand)), X)

curhi = sum(np.random.multivariate\_normal(mean = B\_curr, cov = XtWX))

lasthi = sum(np.random.multivariate\_normal(mean = B\_cand, cov = XtWX))

r = (LL\_cand - LL\_curr) + (curgi - lastgi) - (curhi - lasthi)

if r > MHT:

i += 1

u = r - 1.0

else:

u = np.random.uniform(0,1)

u = np.log(u)

if u < r:

B\_curr = B\_cand

LL\_curr = LL\_cand

return(B\_curr)

B\_fin = RandBeta(X, y, B, LL\_curr, knots)

B\_fin