Multivariable Fractional Polynomials with Extensions

Edwin Kipruto Michael Kammer Patrick Royston Willi Sauerbrei

June 30, 2023

Contents

1 Introduction to Multivariable Fractional Polynomial(MFP)

1.1 Overview of MFP

Multivariable regression models are widely used across various fields of science where empirical data is analyzed. In model building, many researchers often assume a linear function for continuous variables, sometimes after applying "standard" transformations such as logarithmic, or divide the variable into several categories. Assuming linearity without considering non-linear relationships may hinder the detection of stronger effects or even cause the effects to be mis-modeled. Categorizing continuous variables, which results in modeling implausible step functions, is a common practice but widely criticized (Royston et al. 2006; Sauerbrei et al. 2023).

When building a descriptive model with the aim of capturing the data structure parsimoniously, two components are often considered: Variable selection to identify the subset of "important" predictors that have a significant impact on the outcome and investigation of non-linear relationships between continuous predictors and the outcome.

The MFP approach has been proposed as a pragmatic method that combines variable and function selection simultaneously in multivariable linear regression modeling. This approach retains continuous predictors as continuous, identifies non-linear functions if sufficiently supported by the data, and eliminates weakly influential predictors using backward elimination (BE). Despite its simplicity and ease of understanding for researchers familiar with regression models, the selected models often capture the essential information from the data. The MFP models are relatively straightforward to interpret and report, which is important for transportability and practical usability. In summary, the MFP procedure combines:

- variable selection through backward elimination with
- selection of fractional polynomial (FP) functions for continuous variables

The analyst must decide on nominal significance levels (α_1, α_2) for both components. The choice of these two significance levels has a strong influence on the complexity of the final model. Often, the same significance levels $(\alpha_1 = \alpha_2)$ are used for both components, but they can also differ. The decision regarding these significance levels strongly depends on the specific aim of the analysis.

The rest of the paper is organized as follows. Section 1.2 provides an overview of fractional polynomial functions for a single continuous covariate in the model, including the function selection procedure (FSP). Section 1.3 describes the MFP approach, focusing on models involving two or more covariates.

Section 2 serves as an introduction to the mfp2 package. It covers the installation process and provides instructions for utilizing the package in various linear regression models. The package's functionality is predominantly demonstrated using Gaussian linear models (Section 2.4), while other models, such as logistic (Section 2.5), Poisson (Section 2.6), and Cox (Section 2.7), are briefly explained.

Section 3 introduces an extension of MFP using the approximate cumulative distribution (ACD) transformation of a continuous covariate. This extension allows for modeling a sigmoid relationship between covariates and an outcome variable (subsection 3.1) and may help mitigate the influence of extreme covariate values on a selected function (subsection 3.2).

Lastly, Section 4 describes an additional extension of MFP that is currently not implemented in the `mfp2` package, such as interactions between variables. However, this extension is available in the Stata software.

For more comprehensive information about MFP and its extensions, please visit MFP website (click here).

1.2 Fractional polynomial models for a continuous variable

Suppose that we have an outcome variable, a single continuous covariate x, and a regression model relating them. A starting point is the straight-line model, $\beta_1 x$ (for simplicity, we suppress the constant term, β_0). Often, a straight line is an adequate description of the relationship, but other models should be investigated for possible improvements in fit. A simple extension of the straight line is a power transformation model, $\beta_1 x^p$. The latter model has often been used by practitioners in an ad hoc way, utilizing different choices of `p`. Royston and Altman (1994) formalized the model by calling it a first-degree fractional polynomial or FP1 function. The power `p` is chosen from a pragmatically restricted set of eight elements: $S = \{-2, 1, 0.5, 0, 0.5, 1, 2, 3\}$, where x^0 denotes natural logarithm of x, $log(x)$.

As with polynomial regression, extension from one-term FP1 functions to more complex and flexible two-term FP2 functions is straightforward. The quadratic function $\beta_1 x^1 + \beta_2 x^2$ is written as $\beta_1 x^{p1} + \beta_2 x^{p2}$ in FP terminology. The powers $p1 = 1$ and $p2 = 2$ are members of set `S`. Royston and Altman extended the class of FP2 functions with different powers to cases with equal powers ($p1 = p2 = p$) by defining them as $\beta_1 x^p + \beta_2 x^p log(x)$. These are known as repeated-powers functions. Detailed definition of FP functions or models is given in Section 4.3.1 of Royston and Sauerbrei (2008). For formal definitions, we use notation from Royston and Sauerbrei (2008).

FP1 functions are always monotonic and those with power $p < 0$ have an asymptote as $x \to \infty$. FP2 functions may be monotonic or unimodal (i.e., have one maximum or one minimum for some positive values of x), and they have an asymptote as $x \to \infty$ when both $p1$ and $p2$ are negative. For more details, see Royston and Sauerbrei (2008), Section 4.4. Figure 1 shows FP1 and some FP2 curves. The subset of FP2 powers is chosen to illustrate the flexibility available with a few pairs of powers $(p1, p2)$.

In total, there are 44 models available within the set of FP powers (S), consisting of 8 FP1 models and 36 FP2 models. Although the allowed class of FP functions may seem limited, it encompasses a wide range of diverse shapes. This is illustrated in Figure 1, with the left panel displaying eight FP1 powers and the middle panel depicting a subset of FP2 powers.

Based on extensive experience with real data and several simulation studies, FP1 and FP2 are generally considered adequate in the context of multivariable model building, particularly when variable selection and functional forms is required. The content of this article has been previously published in two encyclopedia articles (Sauerbrei and Royston, 2011; Sauerbrei and Royston, 2016).

1.2.1 Function selection procedure (FSP)

Choosing the best FP1 or FP2 function by grid search, minimizing the deviance (minus twice the maximized log-likelihood), is straightforward. However, having a suitable default function is important for increasing the parsimony, stability, and general usefulness of selected functions. In most of the algorithms implementing fractional polynomial (FP) modeling, the default function is linear-arguably, a natural choice. Therefore, unless the data support a more complex FP function, a straight line model is chosen.

There are occasional exceptions; for example, in modeling time-varying regression coefficients in the Cox model, Sauerbrei et al. (2007) chose a default time (t) transformation of $(log(t))$ rather than (t). It can be assumed that deviance difference between an FPm and an FP \(m-1\) model is distributed approximately as central χ^2 on 2 degrees of freedom (d.f.) (Royston and Sauerbrei 2008, Chapter 4.9; Ambler and Royston (2001). To select a specific function, a closed test procedure (other procedures had been proposed before) was proposed (Royston and Sauerbrei 2008, Section 4.10). The complexity of the finally chosen function is predicated on preliminary decisions as to the nominal significance level (α) and the degree (m) of the most

complex FP model allowed. Typical choices are $\alpha = 0.05$ and FP2 (m = 2). We illustrate the strategy for m = 2, which runs as follows:

1. Test the best FP2 model for x at the (α) significance level against the null model using 4 d.f. If the test is not significant, stop and conclude that the effect of x is "not significant" at the α level. Otherwise continue.
2. Test the best FP2 for x against a straight line at the α level using 3 d.f. If the test is not significant, stop, the final model being a straight line. Otherwise continue.
3. Test the best FP2 for x against the best FP1 at the α level using 2 d.f. If the test is not significant, the final model is FP1, otherwise, the final model is FP2. This marks the end of procedure.

The test at step 1 is of overall association of the outcome with x. The test at step 2 examines the evidence for nonlinearity. The test at step 3 chooses between a simpler or more complex nonlinear model.

1.3 Multivariable fractional polynomial (MFP) procedure

When developing a multivariable model with a relatively large number of candidate covariates (say 20, we are not envisaging the case of high-dimensional data), an important distinction is between descriptive, predictive and explanatory modelling (Shmueli, 2010). MFP was mainly developed for descriptive modelling, aiming to capture the data structure parsimoniously. Nevertheless, a suitable descriptive model often has a fit similar to a model whose aim is good prediction. In some fields, the term explanatory modelling is used exclusively for testing causal theory. Unlike developing a predictive model based on acceptable statistical criteria, developing a model suitable for description is much more challenging (Sauerbrei et al., 2015).

In many areas of science, the main interest often lies in the identification of influential variables and determination of appropriate functional forms for continuous variables. Often, linearity is presumed without checking this important assumption, and much better-fitting nonlinear functions may not be considered. The MFP procedure was proposed as a pragmatic strategy to investigate whether nonlinear functions can improve the model fit (Royston and Sauerbrei, 2008; Sauerbrei and Royston, 1999). MFP combines backward elimination for the selection of variables with a systematic search for possible nonlinearity by the function selection procedure. The extension is feasible with any type of regression model to which BE is applicable. When developing models for description, it is important to consider factors such as model stability, generalizability, and practical usefulness. The philosophy behind MFP modeling is to create interpretable and relatively simple models (Sauerbrei et al. 2007). Consequently, an analyst using MFP should be less concerned about failing to include variables with a weak effect or failing to identify minor curvature in a functional form of a continuous covariate. Modifications that may improve MFP models are combination with post-estimation shrinkage (Dunkler et al., 2016, R-package shrink) and a more systematic check for overlooked local features (Binder and Sauerbrei, 2010, currently not implemented in mfp2 package). Successful use of MFP requires only general knowledge about building regression models.

Two nominal significance level values are the main tuning parameters: α_1 for selecting variables with BE (in the first step of the FSP) and α_2 for comparing the fit of functions within the FSP. Often, $\alpha_1 = \alpha_2$ is a good choice. If available, subject-matter knowledge should replace or at least guide data-dependent model choice. Only minor modifications are required to incorporate various types of subject-matter knowledge into MFP modeling. For a detailed example, see Sauerbrei and Royston (1999). Recommendations for practitioners of MFP modeling are given in Sauerbrei et al. (2007b) and in Royston and Sauerbrei (2008, Section 12.2).

1.3.1 MFP – Key Issues and Approaches to Handling Them

Mainly focusing on the FP component, we briefly mention key issues of MFP modeling and refer to the literature for further reading. Regarding variable selection, we have summarized relevant issues and provided arguments for backward elimination as our preferred strategy (Royston and Sauerbrei 2008, Chapter 2). Even when a search for model improvement using a nonlinear function is not considered, that is, all functions are

assumed linear, it is infeasible to derive a suitable and stable model for description in small datasets. Below we provide some information about sample size needed, but implicitly we assume that the sample size is "sufficient".

1.3.1.1 The variable has to be positive The class of FP1 and FP2 functions includes a log and other transformations which require that the continuous variable must be positive. A preliminary origin-shift transformation can be applied (Royston and Sauerbrei 2008, Chapters 4.7 and 11). For variables with a "spike" of probability mass at zero, a binary indicator variable may be added to the model and the FSP may be modified accordingly (Royston et al., 2010; Becher et al., 2012; Lorenz et al., 2018).

1.3.1.2 Sample size and influential observations All statistical models are potentially adversely affected by influential observations or "outliers". However, compared with models that comprise only linear functions, the situation may be more critical for FP functions because logarithmic or negative power transformations may produce extreme functional estimates at small values of x. Conversely, the same may happen with large positive powers at large values of x. Such transformations may create influential observations that may affect parts of the FSP. To mitigate the impact of influential observations, it is important to assess the robustness of FP functions. Some suggestions for investigating influential points (IPs) and handling such issues in MFP modeling can be found in Royston and Sauerbrei (2008, Chapters 5 and 10) and their paper on improving the robustness of FP models (Royston and Sauerbrei, 2007). Using synthetic data, a more detailed investigation of IPs is given in Sauerbrei et al. (2023). The authors conclude that for smaller sample sizes, IPs and low power are important reasons that the MFP approach may not be able to identify underlying functional relationships for continuous variables and selected models might differ substantially from the true model. However, for larger sample sizes (about 50 or more observations per variable) a carefully conducted MFP analysis is often a suitable way to select a multivariable regression model which includes continuous variables.

1.3.1.3 Local features: Unlike splines which have a local interpretation of the fitted function, FPs provide a curve with a global interpretation. To investigate possible "overlooked" local features of an FP function, Binder and Sauerbrei (2010) conducted a systematic analysis of model fits obtained by MFP. If local features are detected by their MFP + L procedure, statistically significant local polynomials are then parsimoniously added to the predictor from MFP. This enables the identification and incorporation of local features that may have been missed by the global FP function. This approach is not currently implemented in the `mfp2` package, but it is a potential area for future extensions.

2 Introduction to mfp2 package

`mfp2` is a package that selects the MFP model. In addition, it has the ability to model a sigmoid relationship between `x` and an outcome variable `y` using the ACD transformation proposed by Royston (2016). The package offers three options for variable and function selection: p-value, Akaike information criterion (AIC), and Bayesian information criterion (BIC). Furthermore, it provides functions for prediction and plotting. Currently, the package implements linear, logistic, Poisson, and Cox regression models. However, the package is designed in such a way that it can easily incorporate other generalized linear models or parametric survival models.

The main function, `mfp2()`, implements both MFP and MFP with ACD transformation. It offers two interfaces for input data. The first interface allows direct input of the predictor matrix `x` and the outcome vector `y`. The second interface uses a formula object in conjunction with a data.frame, similar to the `glm()` function with slight modifications. Both interfaces are equivalent in terms of functionality.

The authors of `mfp2` are Edwin Kipruto, Michael Kammer, Patrick Royston, and Willi Sauerbrei, with contribution from Gregory Steiner and Georg Heinze. The R package is maintained by Edwin Kipruto, while the STATA version of `mfp` is maintained by Patrick Royston.

This vignette describes basic usage of `mfp2` in R. There are additional vignettes available that will further enhance your understanding on the mfp2 package.

- "Introduction to Multivariable Fractional Polynomial" provides an overview of multivariable fractional polynomials.

2.1 Estimation algorithm

The estimation algorithm employed in `mfp2` sequentially processes the predictors using a back-fitting approach. It calculates the p-values of each predictor using the likelihood ratio test, assuming linearity. Subsequently, the predictors are arranged based on these p-values. By default, the predictors are arranged in order of decreasing statistical significance. This ordering aims to prioritize modeling relatively important variables before less important ones. This approach may help mitigate potential challenges in model fitting arising from collinearity or more generally, the presence of "concurvity" among the predictors (stata??). Although alternative options for predictor ordering are available, we prefer the default option.

If a predictor contains nonpositive values, the program by default shifts the location of the predictor x to ensure positivity. In addition, it scales the shifted predictor before the first cycle of the algorithm. For more information on shifting and scaling, please refer to section xx.

At the initial cycle, the best-fitting FP function for the first variable (after ordering) is determined, with all the other variables assumed to be linear. The functional form (but not the estimated regression coefficients) is kept, and the process is repeated for the other variables. The first iteration concludes when all the variables have been processed in this way. The next cycle is similar, except that the functional forms from the initial cycle are retained for all variables except the one currently being processed

A variable whose functional form is prespecified to be linear is tested for exclusion within the above procedure when its nominal p-value is less than 1 or argument `keep = FALSE`; otherwise, it is included. Updating of FP functions and candidate variables continues until the functions and variables included in the overall model do not change (convergence). Convergence is usually achieved within 1–5 cycles.

2.2 Installation

To install the `mfp2` package, enter the following command in the R console:

```
install.packages("mfp2")
```

2.3 Quick Start

The purpose of this section is to provide users with a comprehensive understanding of the `mfp2` package. We will provide a concise overview of its key functions and resulting outputs. We will delve into each function in detail, highlighting their specific use. This will provide users with a deeper understanding of the package's functionality. The package includes a built-in dataset that is specifically designed for analysis within the `mfp2` framework.

To begin, let's load the `mfp2` package:

```
library(mfp2)
```

2.4 Linear Regression

The default `family` in `mfp2` package is `Gaussian`, which fits a Gaussian linear model. In this section, we will demonstrate how to fit this model. We will use the prostate cancer data (Stamey et al., 1989) included in our package. The dataset contains seven predictors (six continuous variables and one binary variable) and a continuous outcome variable (log prostate-specific antigen (lpsa)) of 97 patients with prostate cancer. Our aim is to determine whether non-linear functional relationships exist between the predictors and the outcome variable.

Load the `prostate` dataset from the `mfp2` package and display the first few rows of the dataset

```
# Load prostate data
data("prostate")
head(prostate)
#> # A tibble: 6 x 9
#>   obsno   age   svi pgg45 cavol weight   bph    cp   lpsa
#>   <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>  <dbl>
#> 1     1    50     0     0 0.560   16.0  0.25  0.25 -0.431
#> 2     2    58     0     0 0.370   27.7  0.25  0.25 -0.163
#> 3     3    74     0    20 0.600   14.7  0.25  0.25 -0.163
#> 4     4    58     0     0 0.300   26.6  0.25  0.25 -0.163
#> 5     5    62     0     0 2.12    31.0  0.25  0.25  0.372
#> 6     6    50     0     0 0.350   25.2  0.25  0.25  0.765

# create predictor matrix x and numeric vector y
x <- as.matrix(prostate[,2:8])
y <- as.numeric(prostate$lpsa)
```

The command loads a dataframe from the **R** data archive since the `mfp2` package is already loaded. We create a matrix `x` and a numeric vector `y` from the dataframe.

2.4.1 Fitting MFP Models Using Default and Formula Interface

The default interface of `mfp2()` requires a matrix of predictors `x` and a numeric vector of response `y` for continuous outcomes. If you have one predictor, make sure you convert it into a matrix with a single column.

We demonstrate how to fit the Gaussian linear model using the default and formula interface with default parameters. The formula interface uses the `fp()` function. The two approaches leads to the same results.

```
# default interface
fit_default <- mfp2(x, y)

# formula interface
fit_formula <- mfp2(lpsa ~ fp(age) + svi + fp(pgg45) + fp(cavol) + fp(weight) + fp(bph) + fp(cp), data =
```

The main distinction between the `mfp2()` using the formula interface and `glm()` functions in **R** is the inclusion of the `fp()` function within the formula. The presence of the `fp()` function in the formula indicates that the variables included within it should undergo fractional polynomial (FP) transformation, provided that the degree of freedom (df) is not equal to 1. A df of 1 indicates a linear relationship, which does not require transformation. Note that df is an argument in the `fp()` function. For more details on the fp() function, please refer to section xx

The variable `svi` is a binary variable and is therefore not passed to the `fp()` function. This is because binary or factor variables do not undergo FP transformation. If a binary variable is passed to the `fp()` function,

the program will automatically set the df to 1, treating the variable as linear. However, passing a factor variable to the `fp()` function will result in an error. For more details on how `mfp2` handles factor variables, refer to section XX.

2.4.2 Shifting and Scaling of Predictors

Fractional polynomials are defined only for positive variables due to the use of logarithms and other powers such as square root. Thus, `mfp2()`function estimates shifting factors for each variables to ensure positivity. The function `find_shift_factor()`, used internally by `mfp2`, automatically estimates shifting factors for each continuous variables. The formula used to estimate the shifting factor for a variable, say `x1`, is given by:

$$shift_{x1} = \gamma - min(x1)$$

where $min(x1)$ is the smallest observed value of `x1`, while γ is the minimum increment between successive ordered sample values of `x1`, excluding 0 (Royston and Sauerbrei,2008). The new variable $x1' = x1 + shift_{x1}$ will then be used by `mfp2()` in estimating the FP powers.

For example, to estimate shifting factors for predictor matrix `x` from prostate data in R, you can run the following code:

```
# minimum values for each predictor
apply(x, 2, min)
#>    age    svi  pgg45  cavol weight    bph     cp
#>  41.00   0.00   0.00   0.26  10.75   0.25   0.25

# shifting values for each predictor
apply(x, 2, find_shift_factor)
#>    age    svi  pgg45  cavol weight    bph     cp
#>      0      0      1      0      0      0      0
```

We see that among the continuous variables, only the variable `pgg45` is shifted by a factor of 1, which is attributed to its minimum value being 0. Even though the variable `svi` also has a minimum value of 0, it is not shifted because its a binary variable. The user can manually set the shifting factors for each variable in `mfp2()` function.

If the values of the variables are too large or too small, it is important to scale the variables to reduce the chances of numerical underflow or overflow which can lead to inaccuracies and difficulties in estimating the model. Scaling can be done automatically or by directly specifying the scaling values for each variables so that the magnitude of the `x` values are not too extreme. By default scaling factors are estimated by the program as follows.

After adjusting the location of `x` (if necessary) so that its minimum value is positive, creating `x'` automatic scaling will divide each value of `x'` by 10^p where the exponent `p` is given by

$$p = sign(k) \times floor(|k|) \quad \text{where} \quad k = log_{10}(max(x') - min(x'))$$

The `mfp2()` function uses this formula to scale `x` matrix, and the scaling process is implemented through the `find_scale_factor()` function. The following R code demonstrates the estimation of scaling factors for `x`. From the output below, we see that the variables `age`, `cavol`, and `cp` have scaling factors of 10 each, while the variables `pgg45` and `weight` have scaling factors of 100 each. Each variable will be divided by its corresponding scaling factor. A scaling factor of 1 implies no scaling.

```
# shift x if nonpositive values exist
shift <- apply(x, 2, find_shift_factor)
xnew <- sweep(x, 2, shift, "+")
```

```
# scaling factors
apply(xnew, 2, find_scale_factor)
#>    age    svi  pgg45  cavol weight    bph     cp
#>     10      1    100     10    100      1     10
```

To manually enter shifting and scaling factors, the `mfp2()` function provides the `shift` and `scale` arguments. In the default usage of `mfp2()`, a vector of shifting or scaling factors, with a length equal to the number of predictors, can be provided. In the formula interface, shifting or scaling factors can be directly specified within the`fp()` function. Below is an example to illustrate this:

```
# Default interface
mfp2(x, y, shift = c(0, 0, 1, 0, 0, 0, 0), scale = c(10, 1, 100, 10, 100, 1,
    10))

# Formula interface
mfp2(lpsa ~ fp(age, shift = 0, scale = 10) + svi + fp(pgg45, shift = 1, scale = 100) +
    fp(cavol, shift = 0, scale = 10) + fp(weight, shift = 0, scale = 100) +
    fp(bph, shift = 0, scale = 1) + fp(cp, shift = 0, scale = 10), data = prostate)
```

In the default interface, each variable in the `x` matrix is assigned a shifting and scaling factor based on their respective positions. For instance, the first variable in the column of `x`, which is `age`, is assigned a shifting factor of 0 and a scaling factor of 10. The second variable, `svi`, is assigned a shifting factor of 0 and a scaling factor of 1, and so on.

2.4.3 Setting degrees of freedom for each variable

The degrees of freedom (df) for each predictor (excluding the intercept) are twice the degrees of freedom of the FP. For instance, if the maximum allowed complexity of variable x_1 is a second-degree FP (FP2), then the degrees of freedom assigned to this variable should be 4. The default `df` is 4 for each predictor.

After assigning the default degrees of freedom to each variable, the program proceeds and overrides the default value based on the number of unique values for a given variable. The rules for overriding the default degrees of freedom are as follows:

- If a variable has 2-3 distinct values, it is assigned `df` = 1 (linear).
- If a variable has 4-5 distinct values, it is assigned `df` = min(2, default).
- If a variable has 6 or more distinct values, it is assigned `df` = default.

These rules ensure that the appropriate `df` are assigned to variables. For instance, it is not sensible to fit an FP2 function to a variable with only 3 distinct values.

The following code illustrates how to set different `df` for each variable. In the default interface, the `df` of the binary variable `svi` is explicitly set to 1, while In the formula interface, there is no need to specify the `df`, as the program automatically assigns `df = 1` for binary variables.

If the user attempts to enter `df = 4` for `svi` in the default interface, the program will reset the `df` to 1 and issue a warning.

```
# Default Interface
mfp2(x, y, df = c(4, 1, 4, 4, 4, 4, 4))
```

```
# Formula Interface
mfp2(lpsa ~ fp(age, df = 4) + svi + fp(pgg45, df = 4) + fp(cavol, df = 4) +
    fp(weight, df = 4) + fp(bph, df = 4) + fp(cp, df = 4), data = prostate)
```

If the user does not explicitly assign `df` to variables, the program will automatically assign a `df = 4` to each variable. It is important to note that even if a continuous variable is not passed to the `fp()function in the formula interface, the`df‘ of that variable will still be set to the default value and will later be adjusted based on the unique values. The following three examples are equivalent:

```
# Default Interface
mfp2(x, y)

# Formula Interface
mfp2(lpsa ~ fp(age) + svi + fp(pgg45) + fp(cavol) + fp(weight) + fp(bph) + fp(cp),
    data = prostate)

# Formula Interface but `cp` not passed to the fp() function
mfp2(lpsa ~ fp(age) + svi + fp(pgg45) + fp(cavol) + fp(weight) + fp(bph) + cp,
    data = prostate)
```

2.4.4 Tuning parameters for MFP

The two key components of MFP are variable selection with backward elimination (BE) and function selection for continuous variables through the function selection procedure (FSP). These components require two nominal significance levels: α_1 for variable selection with BE and α_2 for comparing the fit of the functions within the FSP. The choice of these significance levels strongly influences the complexity and stability of the final model. While it is possible to use the same nominal significance level for both components, they can also differ based on the aims of the analysis.

The `mfp2()` function has an argument called `criterion`, which allows the user to specify the criteria for variable and function selection. The default criterion is `pvalue`, which enables the user to set the two nominal significance levels. Please refer to section section 1.4.4.1 for instructions on setting the nominal significance levels.

Information criteria, such as the Akaike information criterion (AIC) and the Bayesian information criterion (BIC), have been proposed for selecting models fitted on the same data. The `mfp2` package offers an alternative approach to variable and function selection by utilizing information criteria. In the MFP framework, each predictor is evaluated univariately while accounting for the other predictors within an overarching backfitting algorithm. This algorithm iteratively assesses each predictor and selects the model (null, linear, FP1, FP2, etc.) with the minimum AIC or BIC for each variable.

The “null” model refers to a model without the predictor of interest. A “linear” model assumes a linear relationship with the outcome variable, while an FP1 model assumes a non-linear relationship using the FP1 function. The `criterion` argument allows users to specify whether they want to use AIC or BIC criteria for both variable and function selection. If AIC or BIC is selected as the criterion, the nominal significance levels set through the `select` and `alpha` arguments will be ignored. For details on using AIC and BIC in variable and function selection, please refer to section 1.4.4.2.

2.4.4.1 Nominal significance levels The `mfp2()` function has the arguments `select` and `alpha` for setting the nominal significance level for variable selection by BE and for testing between FP models of different degrees, respectively. It is important to note that when using these arguments, you should ensure that the `criterion` is set to `"pvalue"` to correctly use the specified significance levels.

For variable selection, a significance level can be set using the `select` argument. A value of 1 (`select = 1`) for all variables forces them all into the model. Setting the nominal significance level to be 1 for a given variable forces it into the model, leaving others to be selected or not. A variable is dropped if its removal leads to a nonsignificant increase in deviance

On the other hand, the `alpha` argument is used to determine the complexity of the selected FP function. A value of 1 (`alpha = 1`) will choose the most complex FP function permitted for a given variable. For example, if FP2 is the most complex function allowed for variable x1, setting alpha = 1 will select the best FP2 function.

The rules for setting `select` and `alpha` are the same as those for setting `df` (see section 1.4.3). The following R codes shows how to set equal nominal significance levels for variable and function selection ($\alpha_1 = \alpha_2 = 0.05$) for each variable and produces identical results. Setting different nominal significance levels is straightforward. Simply replace the value 0.05 with the desired significance level of your choice.

```
# Default Interface
mfp2(x, y, select = rep(0.05, ncol(x)), alpha = rep(0.05, ncol(x)))

# Formula Interface
mfp2(lpsa ~ fp(age, select = 0.05) + svi + fp(pgg45, select = 0.05) + fp(cavol,
    select = 0.05) + fp(weight, select = 0.05) + fp(bph, select = 0.05) + fp(cp,
    select = 0.05), select = 0.05, alpha = 0.05, data = prostate)
```

In the formula interface, binary variables such as `svi` that are not passed in the `fp()` function utilize the global `select` argument. In our example, the global parameter is set to `select = 0.05`. If several binary variables exist in the model, the global parameters will be used for all of them. However, if specific parameters need to be set for individual binary variables, the user can use the `fp()`function. In summary, if a variable is not passed through the `fp` function, it will utilize the global parameters.

Suppose we want to force the variables "age" and "svi" in the model. To achieve this, we have two options:

- Set `select = 1` for `age` and `svi` in the `mfp2()` function.

- Alternatively, we can use the `keep` argument, which resets the nominal significance levels for `age` and `svi` to 1 if they are different from 1. This ensures that these variables are retained in the model.

```
#------Default Interface
# Set select to 1 for age and svi
mfp2(x, y, select = c(1, 1, 0.05, 0.05, 0.05, 0.05, 0.05), alpha = rep(0.05,
    ncol(x)))

# use keep argument
mfp2(x, y, select = c(0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05), alpha = rep(0.05,
    ncol(x)), keep = c("age", "svi"))

#-------Formula Interface
# use fp() function and set select to 1 for age and svi
mfp2(lpsa ~ fp(age, select = 1) + fp(svi, df = 1, select = 1) + fp(pgg45, select = 0.05) +
    fp(cavol, select = 0.05) + fp(weight, select = 0.05) + fp(bph, select = 0.05) +
    fp(cp, select = 0.05), select = 0.05, alpha = 0.05, data = prostate)

# use keep argument
mfp2(lpsa ~ fp(age, select = 0.05) + svi + fp(pgg45, select = 0.05) + fp(cavol,
    select = 0.05) + fp(weight, select = 0.05) + fp(bph, select = 0.05) + fp(cp,
    select = 0.05), select = 0.05, alpha = 0.05, keep = c("age", "svi"), data = prostate)
```

2.4.4.2 Information criterion Instead of using nominal significance levels (α_1, α_2) for variable and function selection, an alternative approach is to directly utilize the AIC or BIC, defined below.

$$AIC = -2\log(L) + 2k$$
$$BIC = -2\log(L) + \log(n) \times k$$

Where $\log(L)$ is the maximum log-likelihood of the fitted model, which measures how well the model fits the data. The parameter k corresponds to the number of estimated parameters in the model (regression estimates and FP powers) and n is the sample size or the number of observations in the dataset. AIC and BIC consider both model fit and complexity, with lower values indicating better-fitting models.

For instance, when selecting the best model for a variable of interest (z) with fixed adjustment variables x1 (with power p3) and x2 (linear), we can compare the AIC and BIC of different models, such as FP2, FP1, linear, and null models. The adjustment models all have the same number of parameters (4, including the intercept if it exists). However, the FP2, FP1, and linear models have additional 4, 2, and 1 parameters, respectively. The total number of parameters for each model type is used in calculating the AIC and BIC. The model with the smallest AIC or BIC is then selected.

In `mfp2` package, this can be achieved by setting the `criterion` argument to either "aic" or "bic" in the `mfp2()` function. Additionally, if there is a need to force certain variables, such as "age" and "svi", into the model, the `keep` argument can be used.

The following R code demonstrates how to implement this approach using both the default and formula interfaces, as well as how to force specific variables into the model:

```
# Default Interface
mfp2(x, y, criterion = "aic", keep = c("age", "svi"))

# Formula Interface
mfp2(lpsa ~ fp(age) + fp(svi, df = 1) + fp(pgg45) + fp(cavol) + fp(weight) + fp(bph) +
    fp(cp), criterion = "aic", keep = c("age", "svi"), data = prostate)
```

2.4.5 Model comparison tests

The FSP in `mfp2()` function compares various models for the variable of interest. For instance, if the most complex allowed FP function is FP2, the FSP will compare the best FP2 model with the null model, the best FP2 model with the Linear model, and the best FP2 model with the best FP1 model, when the `criterion = "pvalue"`. The deviance for each model (NULL, Linear, FP1, and FP2) and their corresponding differences and p-values will be calculated.

When comparing Gaussian models, the `mfp2()` function provides two options for calculating p-values: the F-test and the Chi-square test. The Chi-square test is the default option. For other model families like Cox or logistic regression models, the Chi-square test is used. For more detailed information, please refer to page 23 of the MFP Stata manual paper available at this link: https://www.stata.com/manuals/rfp.pdf

To use the F-test in **R**, we can set the `ftest` argument to TRUE (`ftest = TRUE`), as demonstrated in the example below. Conversely, to use the Chi-square test, set `ftest = FALSE`.

```
# Default Interface
mfp2(x, y, criterion = "pvalue", ftest = TRUE)

# Formula Interface
mfp2(lpsa ~ fp(age) + svi + fp(pgg45) + fp(cavol) + fp(weight) + fp(bph) + fp(cp),
    criterion = "pvalue", ftest = TRUE, data = prostate)
```

Please note that the p-values reported by the `mfp` program in `Stata` for Gaussian models are based on the F-test. If you intend to compare the results between the two software packages, it is crucial to ensure that `ftest = TRUE` is set in R.

However, it is important to be aware that the older version of the `mfp` package in R uses the Chi-square test for all model types. Therefore, when comparing the results between the two R packages, the user must set`ftest = FALSE`.

2.4.6 Fraction Polynomial Powers

Low order polynomials offer a limited family of shapes, and high order polynomials may fit poorly at the extreme values of the covariates. Due to these limitations, Royston an Altman (1994) proposed an extended family of curves known as fractional polynomials, whose power terms are restricted to a small predefined set, $S = \{2, 1, 0.5, 0, 0.5, 1, 2, 3\}$ of integer and non-integer values. The powers are selected so that conventional polynomials are a subset of the family. The power of 0 denotes the natural logarithm of x, log(x), while the power of 1 denotes no transformation. Furthermore, the set includes other powers such as the reciprocal (-1), square root (0.5), and square (2).

By default, the `mfp2()` function utilizes the predefined set S for each continuous covariate. However, there may be situations where users want to provide their own power terms based on their subject matter knowledge. In such cases, the `powers` argument which takes a list of distinct powers can be employed to specify the desired power terms. If the user provides identical powers for a variable, such as (0, 0, 2), the program will remove the duplicates and consider only the powers 0 and 2.

The following example illustrates how to assign different power terms to covariates. Two power terms (0 and 0.5) are evaluated for the "age" covariate. This set includes three linear models of degree 2 (with power combinations of (0, 0), (0, 0.5), and (0.5, 0.5)), and two models of degree 1 (with power of 0 and 0.5).

The "cavol" variable is assigned a single power term (0), indicating that FP1 is the most complex function for this variable. The remaining continuous variables, namely "pgg45," "weight," "bph," and "cp," are not assigned any power terms. Instead, the default powers defined in set S are utilized and a search through all possible fractional polynomials up to the degree set by `df` is performed.

It is important to note that when using the `fp()` function in the formula interface, the power terms provided must be a vector not a list since they are specific to a particular variable.

```
# create a list of power terms for covariates age and cavol
powx <- list(age = c(0, 0.5), cavol = 0)
# Default Interface
mfp2(x, y, criterion = "pvalue", powers = powx)

# Formula Interface
mfp2(lpsa ~ fp(age, powers = c(0, 0.5)) + svi + fp(pgg45) + fp(cavol, powers = 0) +
    fp(weight) + fp(bph) + fp(cp), data = prostate)
```

2.4.7 Explanation of output from model-selection algorithm

Using the prostate example, we briefly explain how the algorithm works. Similar to backward elimination, it starts with the full model (model with all variables) and investigates whether variables can be eliminated. However, for each of the continuous variables, the FSP is used to check whether a non-linear function fits the data significantly better than a linear function. After the first cycle, some variables may be eliminated from the model, and for some continuous variables, a more suitable non-linear function may be identified as a better fit for the data.

The algorithm then proceeds to the second cycle, but the new starting model now has fewer variables due to the elimination process in the previous cycle. Additionally, non-linear functions may have been identified

for some of the continuous variables. In the second cycle, all variables will be reconsidered, even if they were not significant at the end of the first cycle, and FSP is used again to determine the 'best' fitting FP function (the functions may be different due to potential variations in the adjustment variables). The results obtained from the second cycle then serve as the starting model for the third cycle. In most cases, the variables and functions selected remain unchanged in cycles 3 or 4 and the algorithm stops with the final MFP model.

The order of 'searching' for model improvement by better fitting non-linear functions is important. Mismodelling the functional form of a variable with a strong effect is more critical than mismodelling the functional form of a variable with a weak effect. Therefore, the order is determined by the p-values from the full model. Variables with small p-values are considered first.

To explain the output of the MFP algorithm, we will use the default interface of the `mfp2()` function to build an MFP model with the default parameters. Specifically, all continuous variables are assigned a degree of freedom (df) of 4, implying that FP2 is the most complex permitted function. Moreover, we employ the 'pvalue' as the criterion for variable and function selection, and we set the nominal significance levels for both components to 0.05 for all variables. Finally, we use the F-test instead of Chi-square to calculate the p-values. The **R** output displays the df used for each variable, as denoted by the "initial degrees of freedom". The program correctly identifies "svi" as a binary variable and assigns it a df of 1. Additionally, the variables are ordered based on their p-values in descending order of significance, as indicated by the "visiting order". The variable "cavol" has the smallest p-value and will be evaluated first, while "age" has the largest p-value and will be evaluated last.

```
fit <- mfp2(x, y, criterion = "pvalue", select = 0.05, alpha = 0.05, ftest = TRUE)
#>
#> i Initial degrees of freedom:
#>    age svi pgg45 cavol weight bph cp
#> df   4   1     4     4      4   4  4
#>
#> i Visiting order: cavol, svi, pgg45, weight, bph, cp, age
#>
#> ----------------------
#> i Running MFP Cycle 1
#> ----------------------
#>
#> Variable: cavol (keep = FALSE)
#>                 Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2             -0.5, 1   12    196.3       .               .              .
#> null            NA        8     240.1       FP2             43.8           0.0000
#> linear          1         9     214.3       FP2             18.0           0.0011
#> FP1             0         10    199.7       FP2             3.4            0.2226
#> Selected: FP1
#>
#> Variable: svi (keep = FALSE)
#>                 Powers    DF    Deviance    Versus          Deviance diff. P-value
#> null            NA        8     208.6       .               .              .
#> linear          1         9     199.7       null            -9.0           0.0042
#> Selected: linear
#>
#> Variable: pgg45 (keep = FALSE)
#>                 Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2             -2, -2    12    196.7       .               .              .
#> null            NA        8     202.0       FP2             5.3            0.3091
#> Selected: null
#>
#> Variable: weight (keep = FALSE)
```

```
#>                  Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2              -2, -2    11    199.3       .               .              .
#> null             NA        7     209.7       FP2             10.4           0.0521
#> Selected: null
#>
#> Variable: bph (keep = FALSE)
#>                  Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2              -1, 3     10    207.7       .               .              .
#> null             NA        6     217.7       FP2             10.0           0.0567
#> Selected: null
#>
#> Variable: cp (keep = FALSE)
#>                  Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2              2, 3      9     213.2       .               .              .
#> null             NA        5     217.9       FP2             4.6            0.3655
#> Selected: null
#>
#> Variable: age (keep = FALSE)
#>                  Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2              -1, -1    8     216.6       .               .              .
#> null             NA        4     217.9       FP2             1.2            0.8839
#> Selected: null
#>
#> ---------------------
#> i Running MFP Cycle 2
#> ---------------------
#>
#> Variable: cavol (keep = FALSE)
#>                  Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2              -0.5, 1   7     215.0       .               .              .
#> null             NA        3     264.6       FP2             49.6           0.0000
#> linear           1         4     238.1       FP2             23.0           0.0001
#> FP1              0         5     217.9       FP2             2.8            0.2646
#> Selected: FP1
#>
#> Variable: svi (keep = FALSE)
#>                  Powers    DF    Deviance    Versus          Deviance diff. P-value
#> null             NA        3     226.9       .               .              .
#> linear           1         4     217.9       null            -9.0           0.0032
#> Selected: linear
#>
#> Variable: pgg45 (keep = FALSE)
#>                  Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2              0.5, 3    8     214.3       .               .              .
#> null             NA        4     217.9       FP2             3.6            0.4973
#> Selected: null
#>
#> Variable: weight (keep = FALSE)
#>                  Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2              -2, -2    8     202.1       .               .              .
#> null             NA        4     217.9       FP2             15.7           0.0052
#> linear           1         5     205.0       FP2             2.9            0.4438
#> Selected: linear
```

```
#>
#> Variable: bph (keep = FALSE)
#>                 Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2             0.5, 0.5  9     199.1       .               .              .
#> null            NA        5     205.0       FP2             5.9            0.2460
#> Selected: null
#>
#> Variable: cp (keep = FALSE)
#>                 Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2             2, 3      9     200.3       .               .              .
#> null            NA        5     205.0       FP2             4.7            0.3617
#> Selected: null
#>
#> Variable: age (keep = FALSE)
#>                 Powers    DF    Deviance    Versus          Deviance diff. P-value
#> FP2             -0.5, 0   9     203.2       .               .              .
#> null            NA        5     205.0       FP2             1.8            0.7926
#> Selected: null
....
```

After the variables are ordered, the MFP algorithm starts by searching for a suitable function for "cavol". It compares the best-fitting FP2 (-0.5, 1) function for "cavol" against a null model that excludes "cavol." This comparison involves adjusting for all other six variables ("svi" to "age") using linear functions. The test is highly significant (p = 0.0000), indicating that the best FP2 function fits significantly better than a null model. Next, the model with the best FP2 function is compared to a model assuming linearity, and the test remains significant (p = 0.0011). This suggests that "cavol" can be better described by a non-linear function at this stage. Finally, the best FP2 function is compared to the best FP1 (0) function, and the test is not significant (p = 0.2226). Thus, at this stage in the model-selection procedure, the final function for "cavol" is FP1 with power 0, denoting a log function.

The next variable to be evaluated is "svi," which is a binary variable. In this case, only the test of null versus linear is appropriate. Both the null and linear models of "svi" include "log(cavol)" (the log function just selected) and the other five variables ("pgg45", "cp", "weight", "bph", and "age") that are still present in the model. The test of null versus linear is significant (p = 0.0042), indicating that "svi" remains in the model.

Next, we evaluate the continuous variable "pgg45" in the same model as for "svi". The p-value for the first test (FP2 versus null) is not significant (p = 0.3091), and the variable is eliminated from the model and will not be considered in the rest of the first cycle.

Following that, the algorithm evaluates the variable "weight" in a model adjusting for "log(cavol)", "svi", and the three variables ("cp", "bph", and "age") that have not yet been evaluated. The first test is non-significant (p = 0.0521), and the variable is eliminated for the rest of the first cycle. In the subsequent steps, "bph", "cp", and "age" are also eliminated.

At the end of the first cycle, only two variables were selected: "log(cavol)" and "svi".

The variables selected in the first cycle becomes the new starting model for the second cycle. In the second cycle, the effect of "cavol" is investigated again, but this time in a simpler model adjusting for "svi" only. Deviances are larger compared to cycle 1 because the five variables ("pgg45", "age", "weight", "bph" and "cp") no longer belong to the 'adjustment' model. However, the FSP still selects "log(cavol)".

The "svi" is evaluated in a model that includes only "log(cavol)", and the test of linear vs null confirms that the variable should be included in the model. Pgg45 is evaluated in a model with "svi" and "log(cavol)" as adjustment variables, but the test of inclusion (FP2 vs null) is not significant and the variable is again eliminated.

The variable "weight" is considered next in a model with "svi" and "log(cavol)". In contrast to the first cycle, the test for inclusion (FP2 vs null) is significant (p = 0.0052), indicating that "weight" needs to be re-included into the model. The second test of the FSP (FP2 vs linearity) is non-significant and a linear function is chosen for weight. The inclusion of weight in the second cycle is a result of eliminating "bph", "cp" and "age" in the first cycle, and of the correlation with these variables

In the subsequent steps, "bph", "cp", and "age" are investigated in models adjusting for "log(cavol)", "svi", and "weight" (linear). Compared to the first cycle, the p-values change, but all of them are much larger than 0.05. Therefore, none of these variables are included in the model.

The second cycle ends with the model consisting of "log(cavol)", "svi" (binary), and "weight" (linear).

This model serves as the starting point for the third cycle (not shown), where all seven variables are investigated again. However, there are no further change in the selected variables or functions occurs, so MFP terminates with the three variables as the second cycle. For more detailed information, please visit the MFP website at For more detailed information, please visit the MFP website.

2.4.8 Methods defined for mfp2

Once you fit an MFP model using the `mfp2()` function, you can apply various methods to the resulting model object to extract information or perform specific tasks. Suppose we fit an MFP model for the Gaussian family with default parameters as follows:

```
fit <- mfp2(x, y)
```

The `fit` is an object of class `mfp2`, which inherits from `glm` and `lm`. If the `family = "cox"`, the fit will inherit from `coxph`. This means that `mfp2` can utilize methods and functions defined for `glm`, `lm` or `coxph`.

To obtain a summary of the final `mfp2` object, you can simply enter the object name (e.g., "fit" in our example) or use the print function. The `print()` function provides a comprehensive summary of the final MFP model, including the parameters used in the MFP model such as shifting and scaling factors, degrees of freedom (denoted by df_initial), nominal significance levels ("select" and "alpha"), and other relevant information.

Furthermore, it shows the final degrees of freedom (denoted by "df_final"), where a df of 0 indicates that a variable was eliminated, which is confirmed by the "selected" column. The "acd" column specifies whether the ACD transformation was conducted for a certain variable, and the "power" column shows the final FP powers for variables, with eliminated variables assigned NA.

```
print(fit)
#> Shifting, Scaling and Centering of covariates
#>         shift scale center
#> cavol       0    10   TRUE
#> svi         0     1   TRUE
#> pgg45       1   100   TRUE
#> weight      0   100   TRUE
#> bph         0     1   TRUE
#> cp          0    10   TRUE
#> age         0    10   TRUE
#>
#> Final Multivariable Fractional Polynomial for y
#>         df_initial select alpha   acd selected df_final power1
#> cavol           4   0.05  0.05 FALSE     TRUE        2      0
#> svi             1   0.05  0.05 FALSE     TRUE        1      1
```

```
#> pgg45           4   0.05  0.05 FALSE    FALSE        0     NA
#> weight          4   0.05  0.05 FALSE     TRUE        1      1
#> bph             4   0.05  0.05 FALSE    FALSE        0     NA
#> cp              4   0.05  0.05 FALSE    FALSE        0     NA
#> age             4   0.05  0.05 FALSE    FALSE        0     NA
#>
#> MFP algorithm convergence: TRUE
#>
#> Call:  glm(formula = y ~ ., family = family, data = data, weights = weights,
#>     offset = offset, x = TRUE, y = TRUE)
#>
#> Coefficients:
#> (Intercept)      cavol.1        svi.1     weight.1
#>      2.3313       0.5402       0.6794       1.4159
#>
#> Degrees of Freedom: 96 Total (i.e. Null);  93 Residual
#> Null Deviance:       127.9
#> Residual Deviance: 47    AIC: 215
```

To display the regression coefficients from the final MFP model, you can use the `coef()` or `summary()` function. It's important to note that the variables in the model have names with dot extensions. This means that a variable with two FP powers (FP2) will be denoted as "var.1" and "var.2". Please note that the displayed regression coefficients are not currently scaled back to the original scale.

```
# extract only regression coefficients
coef(fit)
#> (Intercept)     cavol.1       svi.1    weight.1
#>   2.3312906   0.5402090   0.6794446   1.4158958

# display regression coefficients with other statistics
summary(fit)
#>
#> Call:
#> glm(formula = y ~ ., family = family, data = data, weights = weights,
#>     offset = offset, x = TRUE, y = TRUE)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -1.7620  -0.4305  -0.0065   0.4730   1.6072
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  2.33129    0.08509  27.399  < 2e-16 ***
#> cavol.1      0.54021    0.07449   7.252  1.2e-10 ***
#> svi.1        0.67944    0.20807   3.265 0.001531 **
#> weight.1     1.41590    0.38967   3.634 0.000458 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for gaussian family taken to be 0.5054218)
#>
#>     Null deviance: 127.918  on 96  degrees of freedom
#> Residual deviance:  47.004  on 93  degrees of freedom
```

```
#> AIC: 215
#>
#> Number of Fisher Scoring iterations: 2
```

Users can generate predictions for new or existing data based on the fitted MFP model using the `predict()` function. To illustrate this, let's consider an example using the prostate dataset. Suppose we want to make predictions for the first five observations in the matrix x (considering them as new observations). We can achieve this with the following code:

```
 # extract the first five observations from 'x'
new_observations <- x[1:5, ]

# make predictions for these new observations.
predict(fit, newdata = new_observations)
#>         1         2         3         4         5
#> 0.9297156 0.8714946 0.9499953 0.7440425 1.8612448
```

To prepare the `newdata` for prediction, the predict() function applies any necessary shifting and scaling based on the factors obtained from the development data. It is important to note that if the shifting factors are not sufficiently large as estimated from the development data, variables in `newdata` may end up with negative values, which can cause prediction errors if non-linear functional forms are used. A warning is given in this case by the function.

2.4.9 Graphical presentation of FP functions

The regression estimates ($\hat{\beta}$) for FP terms provide incomplete information as they only give limited insight into the fitted function for the variable of interest. A more informative approach is to visualize functions for variables with non-linear effects. The mfp2 package offers the `fracplot()` function specifically designed for this purpose.

By providing the mfp2 object and other relevant arguments, `fracplot()` generates plots of the functions for variables along with 95% confidence limits. You can specify the variable to be plotted using the `terms` argument. If `terms` is set to NULL (the default), the function returns a list of plots for all variables included in the model.

`fracplot()` function, by default produces a component-plus-residual plot. This is because the `partial_only` argument is set to FALSE by default.

For Gaussian models with constant weights and a single covariate, this amounts to a plot of the outcome variable (y) and a covariate with the fitted line inscribed.

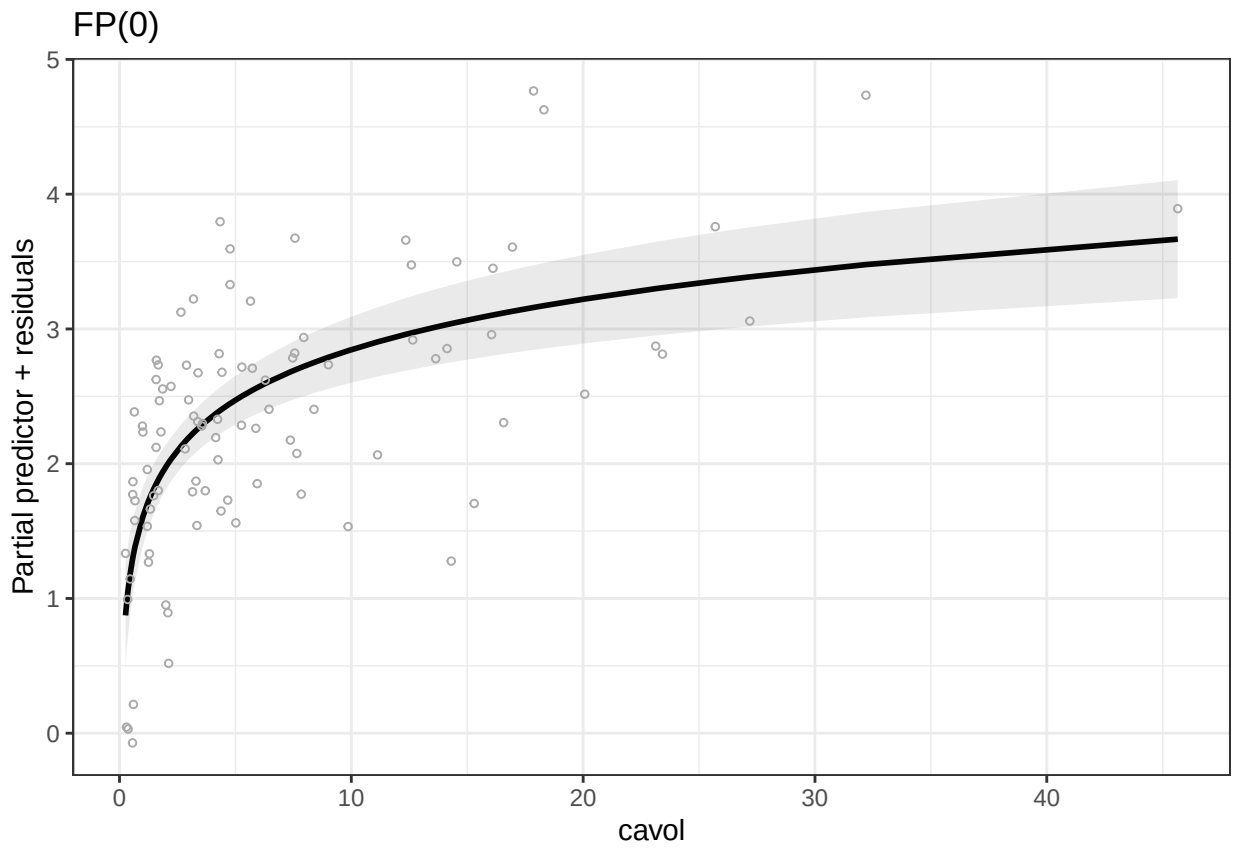
For other normal-error models, weighted residuals are calculated and added to the fit. For models with two or more covariates, the line is the partial linear predictor for the variable of interest and includes the intercept.

For generalized linear and Cox models, the `fracplot()` function plots the fitted values on the scale of the linear predictor. Deviance residuals are added to the (partial) linear predictor for generalized linear models, while martingale residuals are added for Cox models to give component-plus-residual. These values are small circles on the plot.The component-plus-residual plots may show the amount of residual variation at each covariate and may indicate lack of fit or outliers in the outcome (Royston and Sauerbrei, 2008).

If you set `partial_only = TRUE`, the fracplot() function will plot only the partial linear predictor without including the residuals. For more detailed information on component-plus-residuals, you can refer to the Stata manual(click here).

For instance, to plot the function for "cavol" in the model, you can use the following code:

```
plots <- fracplot(fit)
class(plots)
#> [1] "list"
plots[[1]] + ggplot2::ylab("Partial predictor + residuals")
```



The fracplot() function also supports the incorporation of a reference point or target value in the plot. Including a reference point allows for visualizing whether other data points exceed or fall below the reference. To achieve this, you can use the `type` argument and set it to "contrasts" (`type = "contrasts"`). Additionally, you can supply the reference point using the `ref` argument. By default, the `ref` argument is set to `NULL`, and average values are used as the reference points for continuous variables, while the minimum values are used as reference points for binary variables. For instance, if we want the reference value for the continuous variable "cavol" to be 30, we can use the following code:

```
plots <- fracplot(fit, type = "contrasts", ref = list(cavol = 30))
class(plots)
#> [1] "list"
plots[[1]] + ggplot2::ylab("Partial predictor")
```

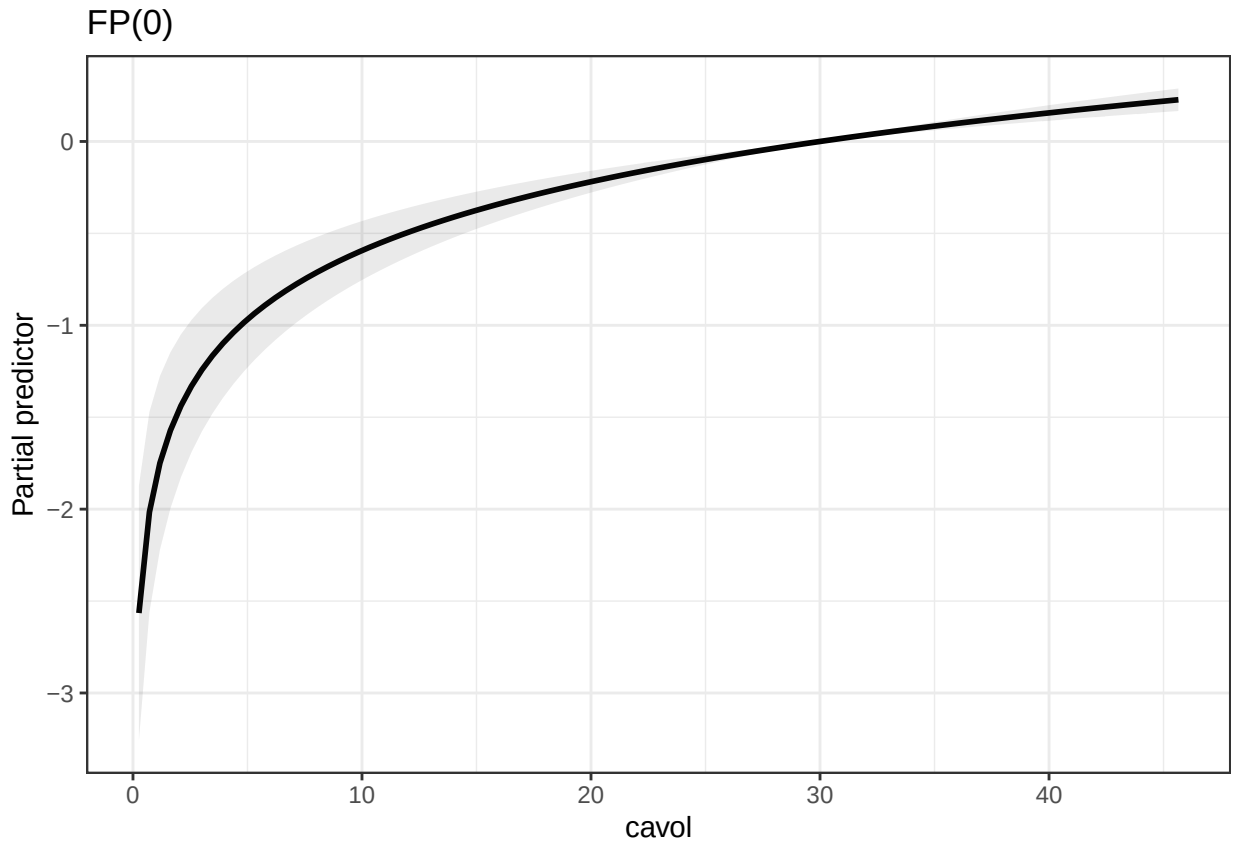


Figure 1: Prostate data. Illustration on how to use reference points

In certain cases, when the data points for a specific variable contain extreme values, the resulting function plotted using the original data can appear very sharp. This sharpness is primarily due to the limited number of data points that are close to the extreme values of the variable.

The `fracplot()` function provides two options for generating the plot. The first option is to use the original data that was used to fit the model (`terms_seq = "data"`). The second option is to generate a sequence of equidistant new data points using the range of the original data for the variable of interest (`terms_seq = "equidistant"`).

To illustrate these options, we will use the `art` dataset included in the mfp2 package. This dataset is described in detail by Royston and Sauerbrei (2008), and the outcome variable is continuous. In our example, we will fit an MFP model with a single covariate (x5).

```r
# load art data
data("art")

# fit mfp model using art data
xx <- as.matrix(art,"x5", drop = FALSE])
yy <- as.numeric(art$y)
fit2 <- mfp2(xx, yy, verbose = FALSE)
#>
#> i Fractional polynomial fitting algorithm converged after 2 cycles.

# generate plot using original data
plot1 <- fracplot(fit2)
plot1[[1]] <- plot1[[1]] + ggplot2::ylab("Partial predictor + residuals")

# generate plot using sequence of data
plot2 <- fracplot(fit2, terms_seq = "equidistant")
plot2[[1]] <- plot2[[1]] + ggplot2::ylab("Partial predictor + residuals")

# combine plots
patchwork::wrap_plots(plot1[[1]], plot2[[1]], ncol = 2, widths = 8)
```

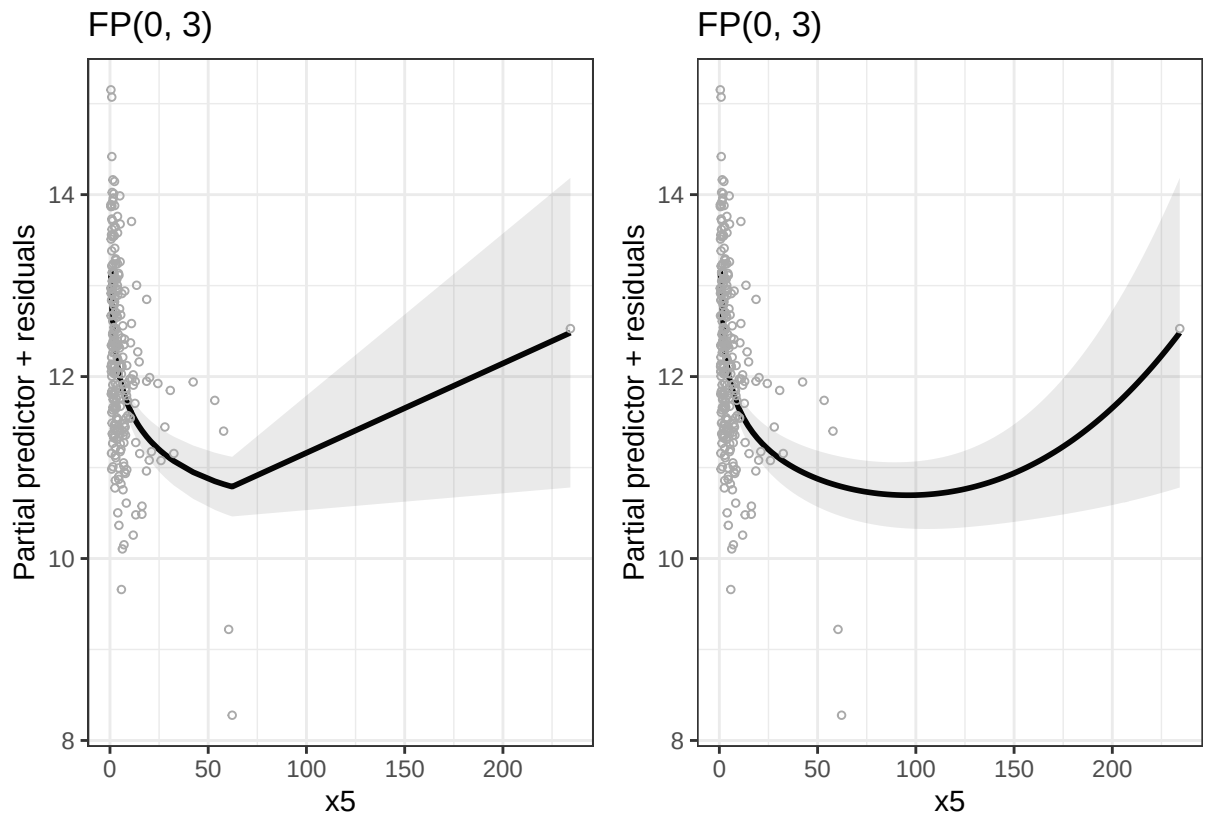


Figure 2: Art data. Illustration on how to use terms_seq argument in fracplot

2.4.10 Handling categorical variables

The `mfp2` package requires the creation of dummy (indicator) variables for qualitative independent variables before using the `mfp2()` function. The number of dummy variables depends on the number of classes in the categorical variable. As a general rule of thumb, a categorical variable with `k` classes is represented by `k-1` dummy variables, each taking on the values of 0 and 1 (Kutner et al., 2005).

Categorical variables can be measured using an ordinal scale or a nominal scale. An ordinal variable indicates that the levels of the variable are ordered in some way, such as the levels of income (e.g., low income, medium income, and high income). On the other hand, a nominal variable has no intrinsic ordering among its categories. When variable selection is involved in the model building process, it is important to consider different approaches for coding ordinal and nominal variables in order to facilitate interpretation. This is because, during variable selection, certain dummy variables may be eliminated, which is analogous to combining some levels of variables. For more details on handling categorical variables, refer to Royston and Sauerbrei (2008, Chapter 3).

The `model.matrix()` function in **R** automatically generates dummy variables for categorical variables. However, it may not be suitable for ordinal variables. Fortunately, the `mfp2` package offers a function called `create_dummy_variables()` that provides an alternative to the `model.matrix()` specifically designed for creating dummy variables for both ordinal and nominal variables.

Before using the `create_dummy_variables()` function, we recommend converting the variable of interest to a factor using the `factor()` function in base R. If necessary, you can specify the levels of the variable. Otherwise, the first level will be used as the reference. We will illustrate how to handle categorical variables using the `art` data. The outcome variable is continuous, and there are 10 covariates. For more details about the data, please refer to Royston and Sauerbrei (2008).

The R code below demonstrates how to convert the ordinal variable x4, which has 3 levels (1 as the reference, 2, and 3), to dummy variables using ordinal coding. Similarly, it demonstrates how to convert the nominal variable x9, which also has 3 levels (1 as the reference, 2, and 3), to dummy variables using categorical coding. The final data will contain the dummy variables, which can be supplied to the `mfp2()` function.

```
# load art data
data("art")
# order the levels of the variable such that Levels: 1 < 2 < 3
art$x4 <- factor(art$x4, ordered = TRUE, levels = c(1, 2, 3))

# convert x9 to factor and assign level 1 as reference group
art$x9 <- factor(art$x9, ordered = FALSE, levels = c(1, 2, 3))

# create dummy variables for x4 and x9 based on ordinal an categorical coding and drop
# the original variable
art <- create_dummy_variables(art, var_ordinal = c("x4"), var_nominal = c("x9"),
                              drop_variables = TRUE)

# display the first 20 observations
head(art, 10)
#>           y x1 x2 x3        x5  x6  x7 x8   x10 x4_1 x4_2 x92 x93
#> 1  12.81324 60  1 24  1.829647 247 302  0 33.13    0    0   0   0
#> 2  11.54249 35  1 20 10.335463  64  97  0 12.75    1    0   0   0
#> 3  10.15125 70  0 21  7.155877   0  76  1 16.92    1    0   0   0
#> 4  11.44530 64  0 17  7.745342  21   2  1 15.12    1    0   0   0
#> 5  13.01152 71  0 20  1.851301 226  92  0 31.75    1    0   1   0
#> 6  12.50528 50  1 17  2.471984  35 108  1  9.08    1    0   0   0
#> 7  11.48337 49  1 13  7.277893  15  66  1 23.75    1    1   0   0
#> 8  11.90820 44  1 20  6.306514  89 115  0 34.99    1    0   0   0
```

```
#> 9  12.55809 55  1 26  6.544373 171 133  1 22.64    1    0   1   0
#> 10 12.53969 56  1 14  2.057520  16 201  0 30.63    1    0   0   0
```

After creating the dummy variables (x4_1 and x4_2 for x4, and x92 and x93 for x9), we fit the MFP model using the `mfp2()` function with default parameters. If a categorical variable is not an important predictor of the outcome, all the dummy variables will be eliminated, as in the case of x9 where all the dummies were eliminated. On the other hand, variable x4 is an important predictor, and only one dummy (x4_2) out of the two was eliminated (similar to combining levels 2 and 3) as shown below.

```
# create matrix x and outcome y
x <- as.matrix(art[,-1])
y <- as.numeric(art$y)

# fit mfp using default interface with default parameters
fit <- mfp2(x,y, verbose = FALSE)
#>
#> i Fractional polynomial fitting algorithm converged after 2 cycles.
print(fit)
#> Shifting, Scaling and Centering of covariates
#>      shift scale center
#> x5       0   100   TRUE
#> x1       0    10   TRUE
#> x7       1  1000   TRUE
#> x10      0    10   TRUE
#> x4_1     0     1   TRUE
#> x93      0     1   TRUE
#> x8       0     1   TRUE
#> x92      0     1   TRUE
#> x4_2     0     1   TRUE
#> x6       1  1000   TRUE
#> x2       0     1   TRUE
#> x3       0    10   TRUE
#>
#> Final Multivariable Fractional Polynomial for y
#>      df_initial select alpha   acd selected df_final power1 power2
#> x5            4   0.05  0.05 FALSE     TRUE        4      0      3
#> x1            4   0.05  0.05 FALSE     TRUE        1      1     NA
#> x7            4   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x10           4   0.05  0.05 FALSE     TRUE        1      1     NA
#> x4_1          1   0.05  0.05 FALSE     TRUE        1      1     NA
#> x93           1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x8            1   0.05  0.05 FALSE     TRUE        1      1     NA
#> x92           1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x4_2          1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x6            4   0.05  0.05 FALSE     TRUE        2      0     NA
#> x2            1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x3            4   0.05  0.05 FALSE     TRUE        1      1     NA
#>
#> MFP algorithm convergence: TRUE
#>
#> Call:  glm(formula = y ~ ., family = family, data = data, weights = weights,
#>     offset = offset, x = TRUE, y = TRUE)
#>
```

```
#> Coefficients:
#> (Intercept)         x5.1         x5.2         x1.1        x10.1       x4_1.1         x8.1
#>     12.3794      -0.6793       0.1973      -0.2169       0.1787      -0.3616       0.3324
#>        x6.1         x3.1
#>      0.2318      -0.2350
#>
#> Degrees of Freedom: 249 Total (i.e. Null);  241 Residual
#> Null Deviance:       245
#> Residual Deviance: 118.6     AIC: 543
```

2.5 Logistic Regression

Logistic regression is a statistical method used to model binary outcomes. The `mfp2()` function implements the logistic regression model using the `family = "binomial"` argument. For illustration, we will use the Pima Indians dataset included in the mfp2 package, which was downloaded from the MFP website(here). The dataset originates from a study conducted on a cohort of 768 female Pima Indians, a group known to be particularly susceptible to diabetes. Out of the 768 individuals, 268 developed diabetes. The dataset consists of a binary outcome variable, denoting the presence (1) or absence (0) of diabetes, and eight covariates. For more information on the dataset, please refer to the book by Royston and Sauerbrei (2008) Chapter 9.7.

In the `mfp2()` function, it is important to note that the response variable, y, should be provided as a binary vector rather than a matrix. The other arguments whether using the "default" or "formula" interface for binomial regression, are nearly identical to those previously explained for the Gaussian family.

```
# load pima data
data("pima")
head(pima)
#> # A tibble: 6 x 10
#>      id pregnant glucose diastolic triceps insulin   bmi diabetes   age     y
#>   <dbl>    <dbl>   <dbl>     <dbl>   <dbl>   <dbl> <dbl>    <dbl> <dbl> <dbl>
#> 1     1        6     148        72      35     126  33.6    0.627    50     1
#> 2     2        1      85        66      29      76  26.6    0.351    31     0
#> 3     3        8     183        64      20     272  23.3    0.672    32     1
#> 4     4        1      89        66      23      94  28.1    0.167    21     0
#> 5     5        0     137        40      35     168  43.1    2.29     33     1
#> 6     6        5     116        74      19      72  25.6    0.201    30     0

# matrix x
x <- as.matrix(pima[,2:9])
# outcome y
y <- as.vector(pima$y)

# fit mfp
fit <- mfp2(x, y, family = "binomial", verbose = FALSE)
#>
#> i Fractional polynomial fitting algorithm converged after 3 cycles.
fit$fp_terms
#>           df_initial select alpha   acd selected df_final power1 power2
#> glucose            4   0.05  0.05 FALSE     TRUE        1      1     NA
#> bmi                4   0.05  0.05 FALSE     TRUE        2     -2     NA
#> pregnant           4   0.05  0.05 FALSE    FALSE        0     NA     NA
#> diabetes           4   0.05  0.05 FALSE     TRUE        1      1     NA
#> age                4   0.05  0.05 FALSE     TRUE        4      0      3
```

```
#> diastolic          4   0.05  0.05 FALSE    FALSE        0      NA      NA
#> triceps            4   0.05  0.05 FALSE    FALSE        0      NA      NA
#> insulin            4   0.05  0.05 FALSE    FALSE        0      NA      NA
```

From the output, it is evident that four continuous variables (glucose(power 1), bmi(-2), diabetes(1) and age (0, 3)) are selected. We can use `fracplot()` function to generate plots for these selected variables, as illustrated below, which produced Figure 3.

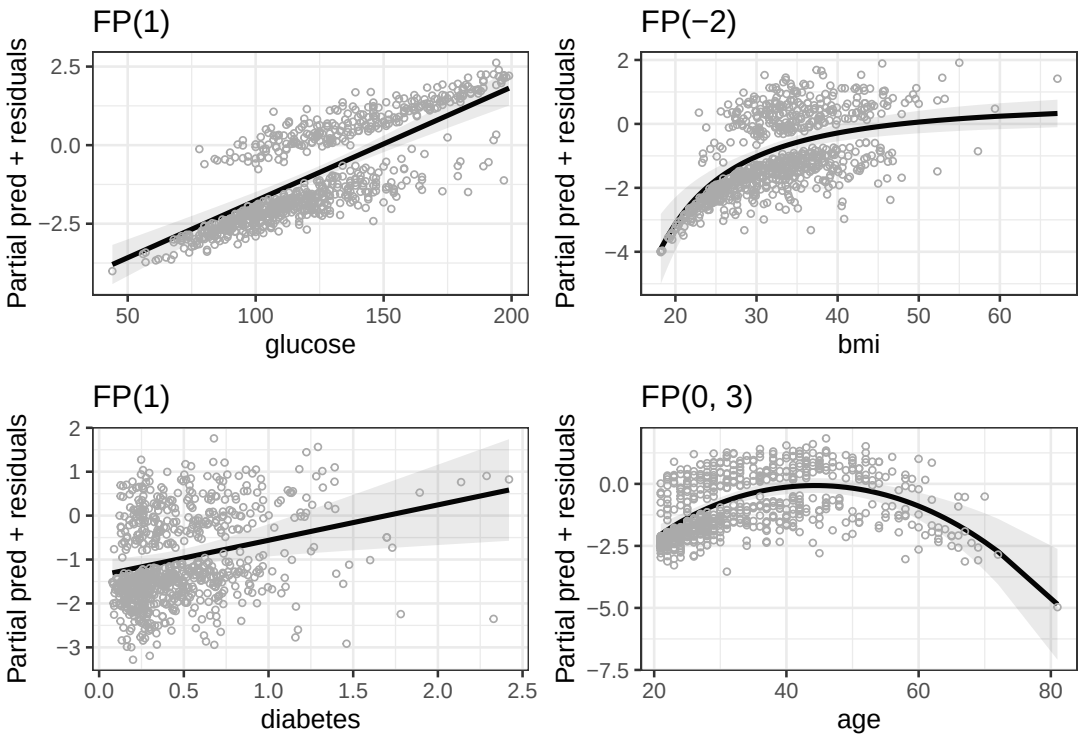


Figure 3: Pima data. Graphical presentation of results from an MFP analysis

2.6 Poisson regression

2.7 Survival data

We illustrate two of the analyses performed by Sauerbrei and Royston (1999). We use the gbsg data, which contains prognostic factors data from the German Breast Cancer Study Group of patients with node-positive breast cancer. The response variable is recurrence-free survival time (rectime), and the censoring variable is censrec. There are 686 patients with 299 events. We use Cox regression to predict the log hazard of recurrence from prognostic factors, of which five are continuous (age, size, nodes, pgr, er), and one is binary (meno). The variable grade is ordinal, and we will use ordinal coding to create two dummy variables (grade_1 and grade_2). Hormonal therapy (hormon) is known to reduce recurrence rates and is forced into the model. Additionally, we stratify the analysis using the hormon variable.

We use the `mfp2()` function with default parameters to build a model from the initial set of eight predictors. We set the nominal significance level for variable and FP selection to 0.05 for all variables except "hormon". "hormon" is forced into the model using the `keep` argument, which sets the significance level of "hormon" to 1.

The `mfp2()` function for survival outcome requires the outcome variable to be a survival object created using the `survival::Surv()` function. Other arguments are similar to those explained previously for the Gaussian family.

```
# load gbsg data
data("gbsg")

#--- data preparation
# create dummy variable for grade using ordinal coding
gbsg <- create_dummy_variables(gbsg, var_ordinal = "grade", drop_variables = TRUE)


# predictor matrix x
x <- as.matrix(gbsg[,-c(1,6, 10,11)])
head(x, 10)
#>       age meno size nodes pgr  er hormon grade_1 grade_2
#>  [1,]  38    0   18     5 141 105      0       1       1
#>  [2,]  52    0   20     1  78  14      0       0       0
#>  [3,]  47    0   30     1 422  89      0       1       0
#>  [4,]  40    0   24     3  25  11      0       0       0
#>  [5,]  64    1   19     1  19   9      1       1       0
#>  [6,]  49    1   56     3 356  64      1       0       0
#>  [7,]  53    1   52     9   6  29      0       1       0
#>  [8,]  61    1   22     2   6 173      1       1       0
#>  [9,]  43    0   30     1  22   0      0       1       0
#> [10,]  74    1   20     1 462 240      1       1       0

# use Surv() function to create outcome y
y <- survival::Surv(gbsg$rectime, gbsg$censrec)

#---fit mfp and keep hormon in the model
fit1 <- mfp2(x, y, family = "cox", keep = "hormon", verbose = FALSE,
             control = coxph.control(iter.max = 50))
#>
#> i Fractional polynomial fitting algorithm converged after 3 cycles.
fit1$fp_terms
#>         df_initial select alpha   acd selected df_final power1 power2
```

```
#> nodes            4   0.05  0.05 FALSE     TRUE        4   -2.0   -1.0
#> pgr              4   0.05  0.05 FALSE     TRUE        2    0.5     NA
#> grade_1          1   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> hormon           1   1.00  0.05 FALSE     TRUE        1    1.0     NA
#> size             4   0.05  0.05 FALSE    FALSE        0     NA     NA
#> meno             1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> grade_2          1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> age              4   0.05  0.05 FALSE     TRUE        4   -2.0   -0.5
#> er               4   0.05  0.05 FALSE    FALSE        0     NA     NA
```

According to Sauerbrei and Royston (1999), medical knowledge dictates that the estimated risk function for nodes (number of positive nodes), which was based on the above FP with powers (-2, -1), should be monotonic, but it was not. They improved Model II by estimating a preliminary exponential transformation, enodes = exp(-0.12 * nodes), for nodes and fitting a degree 1 FP for enodes, thus obtaining a monotonic risk function. The value of -0.12 was estimated univariately using nonlinear Cox regression. To ensure a negative exponent, Sauerbrei and Royston (1999) restricted the powers for enodes to be positive. Their Model III may be fit by using the following R command, which yields identical results to the mfp command in Stata(here):

```
# remove nodes and include enodes
x <- as.matrix(gbsg[,-c(1,5, 10,11)])
#---fit mfp and keep hormon in the model
fit2 <- mfp2(x, y, family = "cox", keep = "hormon", verbose = FALSE,
             powers = list(enodes = c(0.5, 1, 2, 3)), control = coxph.control(iter.max = 50))
#>
#> i Fractional polynomial fitting algorithm converged after 3 cycles.
fit2$fp_terms
#>         df_initial select alpha   acd selected df_final power1 power2
#> enodes           4   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> pgr              4   0.05  0.05 FALSE     TRUE        2    0.5     NA
#> hormon           1   1.00  0.05 FALSE     TRUE        1    1.0     NA
#> grade_1          1   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> size             4   0.05  0.05 FALSE    FALSE        0     NA     NA
#> meno             1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> grade_2          1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> age              4   0.05  0.05 FALSE     TRUE        4   -2.0   -0.5
#> er               4   0.05  0.05 FALSE    FALSE        0     NA     NA
```

2.7.1 stratified Cox model

The stratified Cox model is a variation of the Cox proportional hazards (PH) model that allows for controlling the effect of a predictor that does not satisfy the PH assumption (Therneau et al. 2000). The model includes predictors that are assumed to satisfy the PH assumption, while the predictor being stratified is not included in the model.Instead of including the stratified variable as a covariate in the model, it is used to define distinct subgroups.

The mfp2() function like coxph() function in survival package has a strata argument which allow stratification. The fitted mfp model assumes that the FP functions for a certain variable do not vary over the strata. Obviously, the user should evaluate this assumption

The `mfp2()` function, similar to the `coxph()` function in the survival package, includes a strata argument that enables stratification. The fitted MFP model assumes that the FP functions for a particular variable do not vary over the strata. However, it is essential for the user to evaluate this assumption carefully.

To provide an illustrative example, we will stratify the analysis using the variable "hormon." The following R code demonstrates how to fit a stratified Cox model using both the default and formula interfaces, along with their default parameters.

```
# using default interface
fit2 <- mfp2(x[,-7], y, family = "cox", strata = x[,7],
     control = coxph.control(iter.max = 50),verbose = FALSE)
#>
#> i Fractional polynomial fitting algorithm converged after 2 cycles.
fit2$fp_terms
#>         df_initial select alpha   acd selected df_final power1 power2
#> enodes           4   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> pgr              4   0.05  0.05 FALSE     TRUE        2    0.5     NA
#> grade_1          1   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> size             4   0.05  0.05 FALSE    FALSE        0     NA     NA
#> meno             1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> grade_2          1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> age              4   0.05  0.05 FALSE     TRUE        4   -2.0     -1
#> er               4   0.05  0.05 FALSE    FALSE        0     NA     NA

# using formula interface

fit3 <- mfp2(Surv(rectime,censrec)~ age + meno + size + nodes + pgr + er + grade_1 +
               grade_2 + strata(hormon),family = "cox", data =gbsg, verbose = FALSE)
#>
#> i Fractional polynomial fitting algorithm converged after 3 cycles.

fit2$fp_terms
#>         df_initial select alpha   acd selected df_final power1 power2
#> enodes           4   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> pgr              4   0.05  0.05 FALSE     TRUE        2    0.5     NA
#> grade_1          1   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> size             4   0.05  0.05 FALSE    FALSE        0     NA     NA
#> meno             1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> grade_2          1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> age              4   0.05  0.05 FALSE     TRUE        4   -2.0     -1
#> er               4   0.05  0.05 FALSE    FALSE        0     NA     NA
```

3 MFP with ACD transformation

Royston (2015) proposed the approximate cumulative distribution (ACD) transformation of a continuous covariate x as a route toward modeling a sigmoid relationship between x and an outcome variable. He described an extension of univariate FP modeling via the ACD covariate transformation. The ACD transformation is a smooth function that maps a continuous covariate, x, to an approximation, ACD (x), of its distribution function. By construction, the distribution of ACD (x) in the sample is roughly uniform on (0, 1). FP modeling is then performed with the transformed values ACD (x) instead of x as a predictor. He further showed that such an approach could successfully represent a sigmoid function of x, something a standard FP function cannot do (Royston and Sauerbrei 2008). He went on to demonstrate that useful flexibility in functional form could be achieved by considering both x and a = ACD (x) simultaneously as independent predictors and applying the MFP algorithm to x and a. To limit instability and overfitting, he suggested restricting the models considered for x and a to FP1 functions. Furthermore, Royston (2015) highlighted that models based on ACD (x) may have other advantages in terms of interpretability of regression coefficients

and may help reduce the influence of extreme covariate values on a selected function (see Figure 5 for an example).

Royston and Sauerbrei (2016) extended the MFP modeling approach to incorporate the ACD transformation, resulting in the MFPA (MFP with ACD) transformation. To implement this extension, they introduced the MFPA algorithm, where the original FP FSP is replaced by a modified version known as FSPA (FSP with ACD transformation). The MFPA algorithm has all the features of MFP algorithm plus the ability to model a sigmoid function. For additional information on MFPA, including FSPA, please refer to Royston and Sauerbrei (2016)(click here)

3.1 modeling a sigmoid relationship

We will demonstrate how to fit the MFPA using the `mfp2()` function. In this example, we couldn't find a suitable dataset that exhibits a sigmoid relationship between the outcome variable `y` and predictor variable `x`. Therefore, we will simulate the data to create an example that showcases this relationship. The simulation process is outlined below.

```
# Generate artificial data with sigmoid relationship
set.seed(54)
n <- 500
x <- matrix(rnorm(n), ncol = 1, dimnames = list(NULL, "x") )

# Apply sigmoid transformation to x
sigmoid <- function(x) {
  1 / (1 + exp(-1.7*x))
}

# Generate y with sigmoid relationship to x
y <- as.numeric(20*sigmoid(x) + rnorm(n, mean = 0, sd = 0.2))
```

The `mfp2()` function has an `acdx` argument that allows the user to specify the names of continuous variables to undergo the ACD transformation. The `acdx` invokes the FSPA to determine the best-fitting model (see documentation for more details). The `acdx` can be set in the formula interface using the `fp()` function, as demonstrated in the R code below. To avoid confusion with the original variable `x`, the variable representing the ACD transformation of `x` is named `A(x)`.

```
#-----Fit MFPA
# using default interface
fit <- mfp2(x, y, acdx = "x", verbose = FALSE)
#>
#> i Fractional polynomial fitting algorithm converged after 2 cycles.

# using formula interface
datax <- data.frame(y,x)
fit1 <- mfp2(y ~ fp(x, acdx = TRUE), data = datax, verbose = FALSE)
#>
#> i Fractional polynomial fitting algorithm converged after 2 cycles.

# display selected power terms
fit2$fp_terms
#>         df_initial select alpha   acd selected df_final power1 power2
#> enodes           4   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> pgr              4   0.05  0.05 FALSE     TRUE        2    0.5     NA
```

```
#> grade_1          1   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> size             4   0.05  0.05 FALSE    FALSE        0     NA     NA
#> meno             1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> grade_2          1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> age              4   0.05  0.05 FALSE     TRUE        4   -2.0     -1
#> er               4   0.05  0.05 FALSE    FALSE        0     NA     NA

# fit usual mfp: bad idea but useful for illustration
fit2 <- mfp2(y ~ fp(x, acdx = FALSE), data = datax, verbose = FALSE)
#>
#> i Fractional polynomial fitting algorithm converged after 2 cycles.
fit2$fp_terms
#>   df_initial select alpha   acd selected df_final power1 power2
#> x          4   0.05  0.05 FALSE     TRUE        4      3      3
```

The selected function using the ACD transformation is FP1(1,1), represented by $\beta_0 + \beta_1 x^1 + \beta_2 a^1$. On the other hand, the usual MFP approach selected FP2(3,3), equivalent to $\beta_0 + \beta_1 x^3 + \beta_2 x^3 log(x)$. Figure 4 shows these functions. It is evident from the figure that the ACD approach provides a better fit to the data compared to the usual MFP model, particularly at the extreme ends.

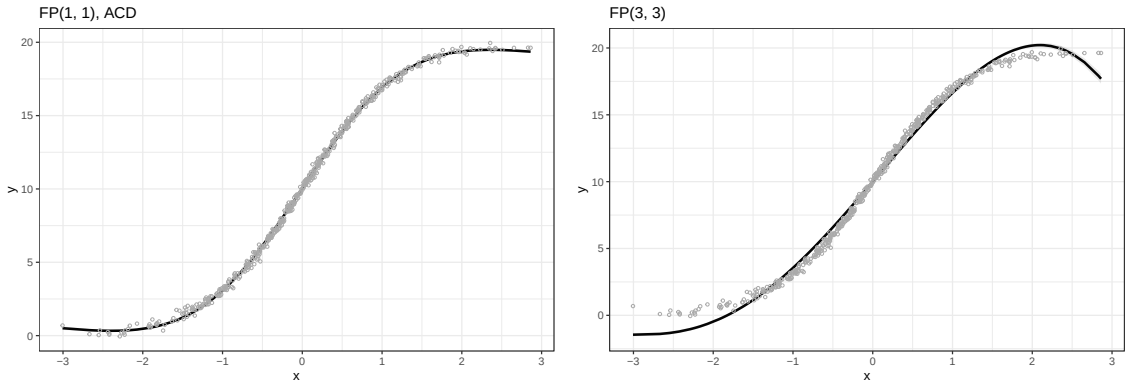


Figure 4: Art data. Relationship between x and outcome y estimated using ACD transformation (left panel) and usual MFP algorithm (right panel)

3.2 use of MFPA to reduce effects of extreme covariate values.

The example below demonstrates the effectiveness of MFPA in mitigating the impact of extreme covariate values on a selected function for variable x5 in the `art` data. We see that the extreme value (observation number 151) has minimal influence on the functional form, compared to usual FP2 as shown in Figure 3.

```
# load art data
data("art")
# create matrix x and outcome y
x <- as.matrix(art[,-1])
y <- as.numeric(art$y)

# fit mfp without acd
fit1 <- mfp2(x,y, verbose = FALSE)
#>
#> ℹ Fractional polynomial fitting algorithm converged after 3 cycles.
fit1$fp_terms
#>     df_initial select alpha   acd selected df_final power1 power2
#> x5           4   0.05  0.05 FALSE     TRUE        4      0      3
#> x1           4   0.05  0.05 FALSE     TRUE        1      1     NA
#> x7           4   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x4           1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x10          4   0.05  0.05 FALSE     TRUE        1      1     NA
#> x9           1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x8           1   0.05  0.05 FALSE     TRUE        1      1     NA
#> x6           4   0.05  0.05 FALSE     TRUE        2      0     NA
#> x2           1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x3           4   0.05  0.05 FALSE     TRUE        1      1     NA

# fit mfp with acd
fit2 <- mfp2(x,y, acdx = "x5", verbose = FALSE)
#>
#> ℹ Fractional polynomial fitting algorithm converged after 3 cycles.
fit2$fp_terms
#>     df_initial select alpha   acd selected df_final power1 power2
#> x5           4   0.05  0.05  TRUE     TRUE        4   -0.5      0
#> x1           4   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> x7           4   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x4           1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x10          4   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> x9           1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x8           1   0.05  0.05 FALSE     TRUE        1    1.0     NA
#> x6           4   0.05  0.05 FALSE     TRUE        2    0.0     NA
#> x2           1   0.05  0.05 FALSE    FALSE        0     NA     NA
#> x3           4   0.05  0.05 FALSE     TRUE        1    1.0     NA

fracplot(fit1, terms = "x5")[[1]] + ggplot2::ylab("Partial predictor + residuals")

fracplot(fit2, terms = "x5")[[1]] + ggplot2::ylab("Partial predictor + residuals")

#patchwork::wrap_plots(list(a,b), ncol = 2)
```

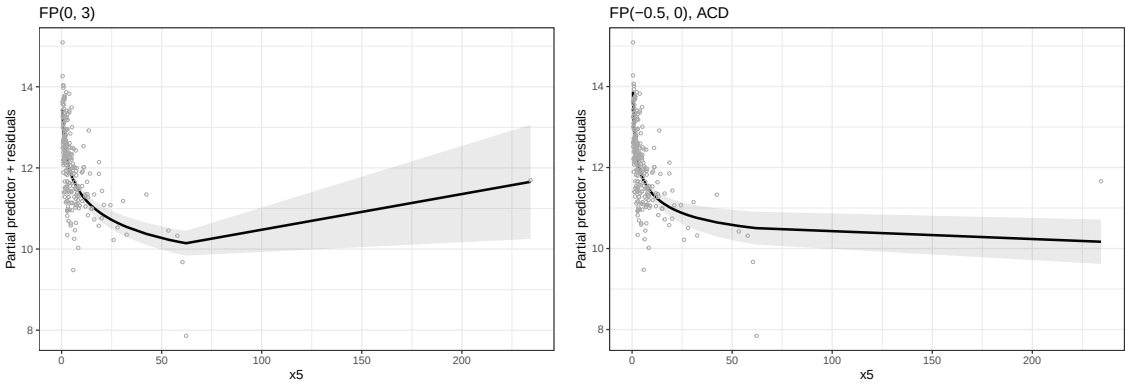


Figure 5: Art data. Effectiveness of MFPA in mitigating the impact of extreme covariate values

3.3 Addition of mfp2 to mfp package