Generalized Additive Model Selection

Trevor Hastie
Department of Statistics
Stanford University

IFCS 2015, Bologna

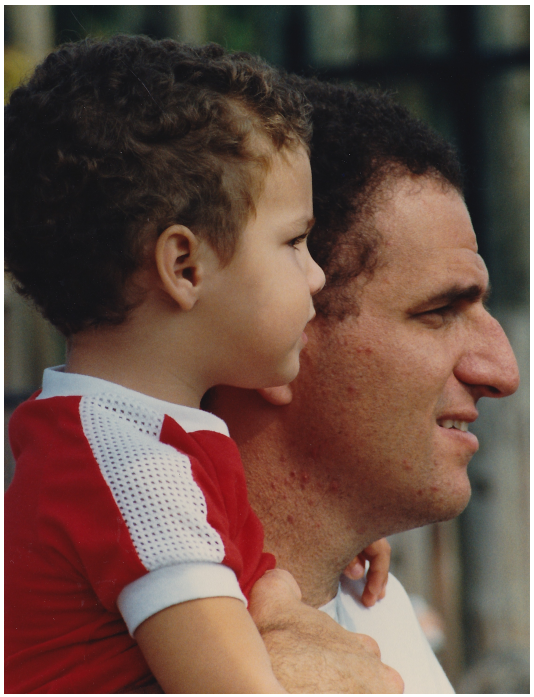


Some History

Three new publications in the late 80s ...

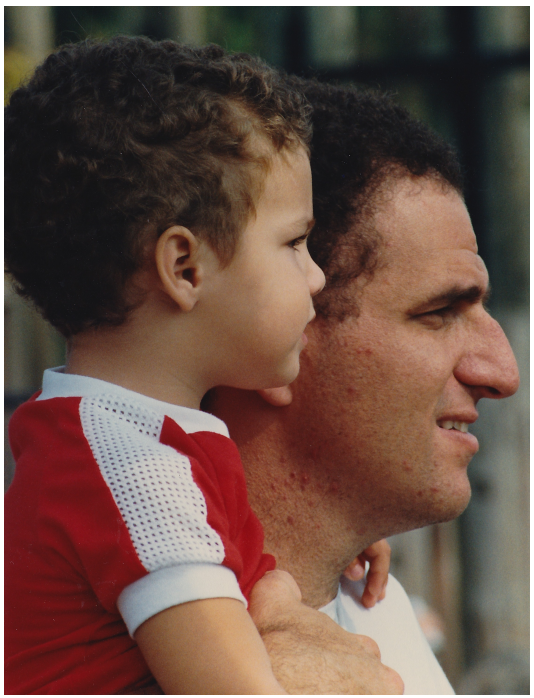
Some History

Three new publications in the late 80s . . .



Some History

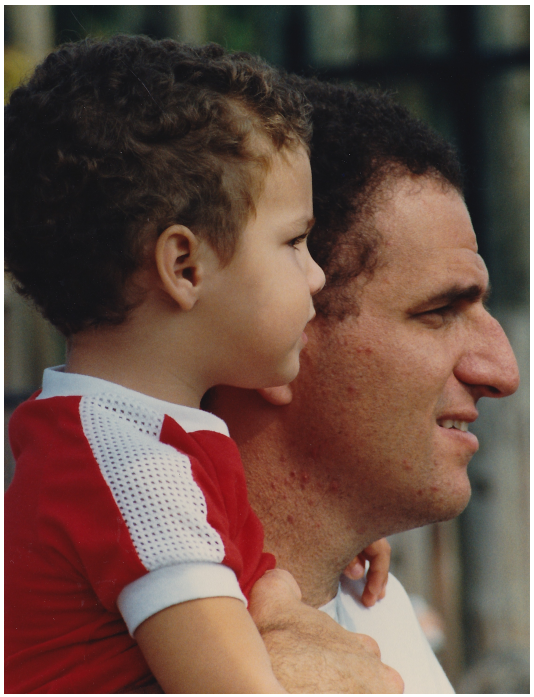
Three new publications in the late 80s . . .



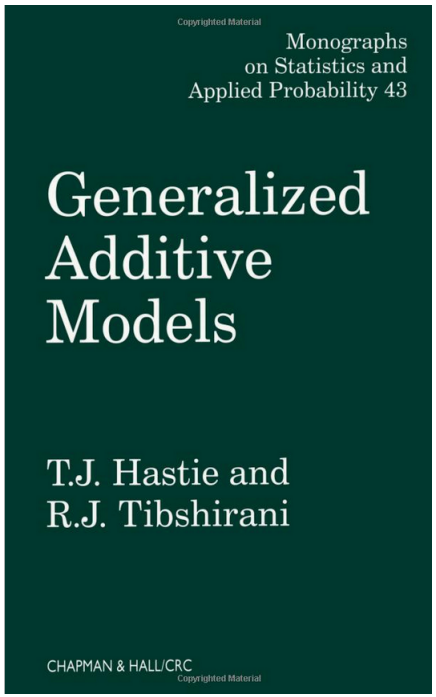


Some History

Three new publications in the late 80s ...

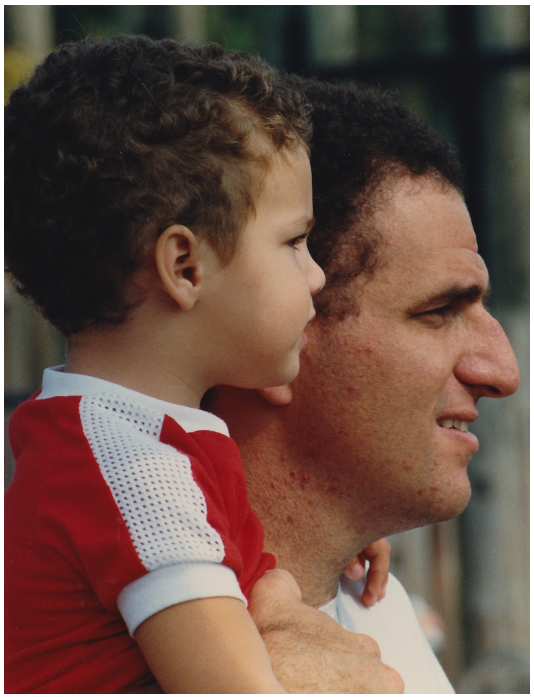




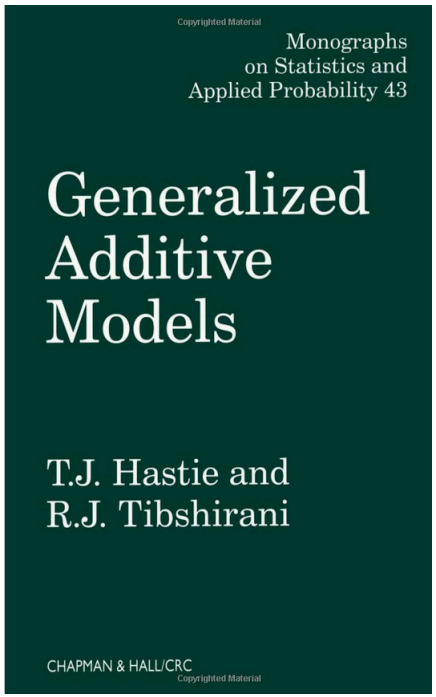


Some History

Three new publications in the late 80s ...







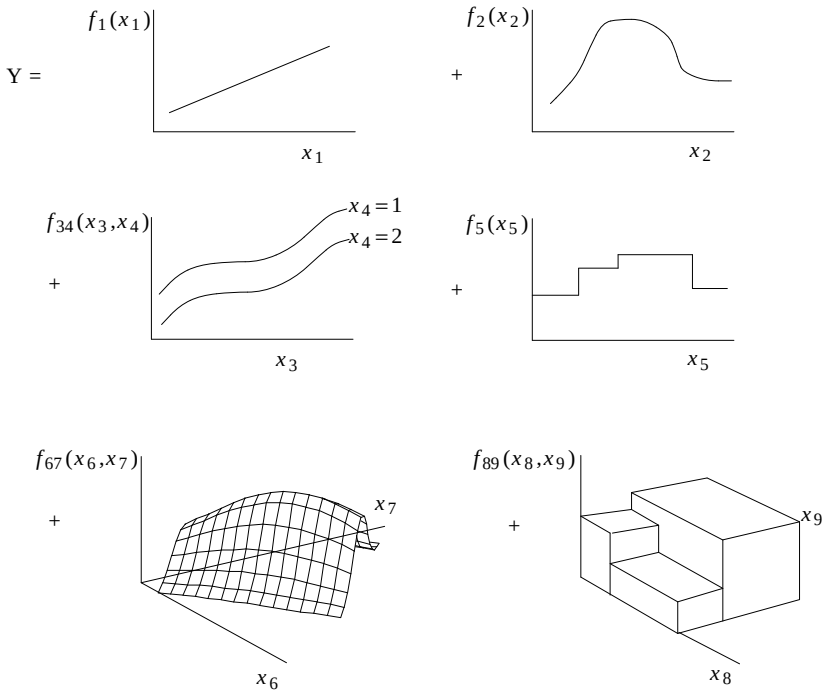
Writing the book got us off diaper duty!

Additive Models

$$Y = \alpha + f_1(x_1) + f_2(x_2) + \cdots + f_p(x_p) + \varepsilon$$

- $Y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$ — linear model
- $Y = \alpha + \{\beta_{11} x_1 + \beta_{12} x_1^2\} + \beta_2 \log(x_2) + \beta_{3j} 1_{(c_j < x_3 \leq c_{j+1})} + \varepsilon$
- $Y = \alpha + \beta_1 x_1 + \beta_2 x_2 + f_3(x_3) + \varepsilon$ — semiparametric model
- $Y = \alpha + \beta_1 x_1 + \beta_2 x_2 + f_3(x_3) + f_4(x_4) + \varepsilon$
- $Y = \alpha + \beta_1 x_1 + f_2(x_2) + f_{34}(x_3, x_4) + \varepsilon$

Example of a (complicated) additive model



Generalized Additive Models

Logistic Regression

$$Y \sim Binomial(1, p(x))$$

$$\begin{aligned} \text{logit } p(x) &= \log\Big(\frac{p(x)}{1-p(x)}\Big) \\ &= \alpha + f_1(x_1) + f_2(x_2) + \cdots + f_p(x_p) \end{aligned}$$

or

$$p(x) = \frac{e^{\alpha+f_1(x_1)+f_2(x_2)+\cdots+f_p(x_p)}}{1+e^{\alpha+f_1(x_1)+f_2(x_2)+\cdots+f_p(x_p)}}$$

$$\text{Var}(Y \mid x) = p(x)(1-p(x))$$

Other Examples

- poisson regression—log link
- transformation models—ACE algorithm
  $g(Y) = \alpha + f_1(x_1) + f_2(x_2) + \cdots + f_p(x_p) + \varepsilon$
- censored survival data—Cox model
  $\lambda(x,t) = \lambda_0(t) e^{f_1(x_1)+f_2(x_2)+\cdots+f_p(x_p)}$
- resistant additive models—tapered likelihoods
- semiparametric additive models for designed experiments
- additive decomposition of time series
- varying coefficient models —
  $\eta(x,t) = \alpha(t) + x_1\beta_1(t) + x_2\beta_2(t)$

Fitting Additive Models

$$Y = f_1(x_1) + f_2(x_2) + \cdots + f_p(x_p) + \varepsilon$$

Estimating Equations

$$\begin{aligned} f_1(x_1) &= S_1(y - \bullet - f_2(x_2) - \cdots - f_p(x_p)) \\ f_2(x_2) &= S_2(y - f_1(x_1) - \bullet - \cdots - f_p(x_p)) \\ &\vdots \\ f_p(x_p) &= S_p(y - f_1(x_1) - f_2(x_2) - \cdots - \bullet) \end{aligned}$$

where S_j are:

- univariate regression smoothers such as smoothing splines, lowess, kernel
- linear regression operators yielding polynomial fits, piecewise constant fits, parametric spline fits, etc
- more complicated operators such as surface smoothers for 2nd order interactions

Justification?

Example: penalized least squares

Minimize $$\sum_{i=1}^{n}\Big(y_i - \sum_{j=1}^{p} f_j(x_{ij})\Big)^2 + \sum_{j=1}^{p}\lambda_j \int (f_j''(t))^2 dt$$

$$\Updownarrow$$

$$\begin{aligned} f_1 &= S_1(\lambda_1)(y - \sum_{j\neq 1} f_j) \\ f_2 &= S_2(\lambda_2)(y - \sum_{j\neq 2} f_j) \\ &\vdots \\ f_p &= S_p(\lambda_p)(y - \sum_{j\neq p} f_j) \end{aligned}$$

where $S_j(\lambda_j)$ denotes a smoothing spline using variable x_j and penalty coefficient λ_j.

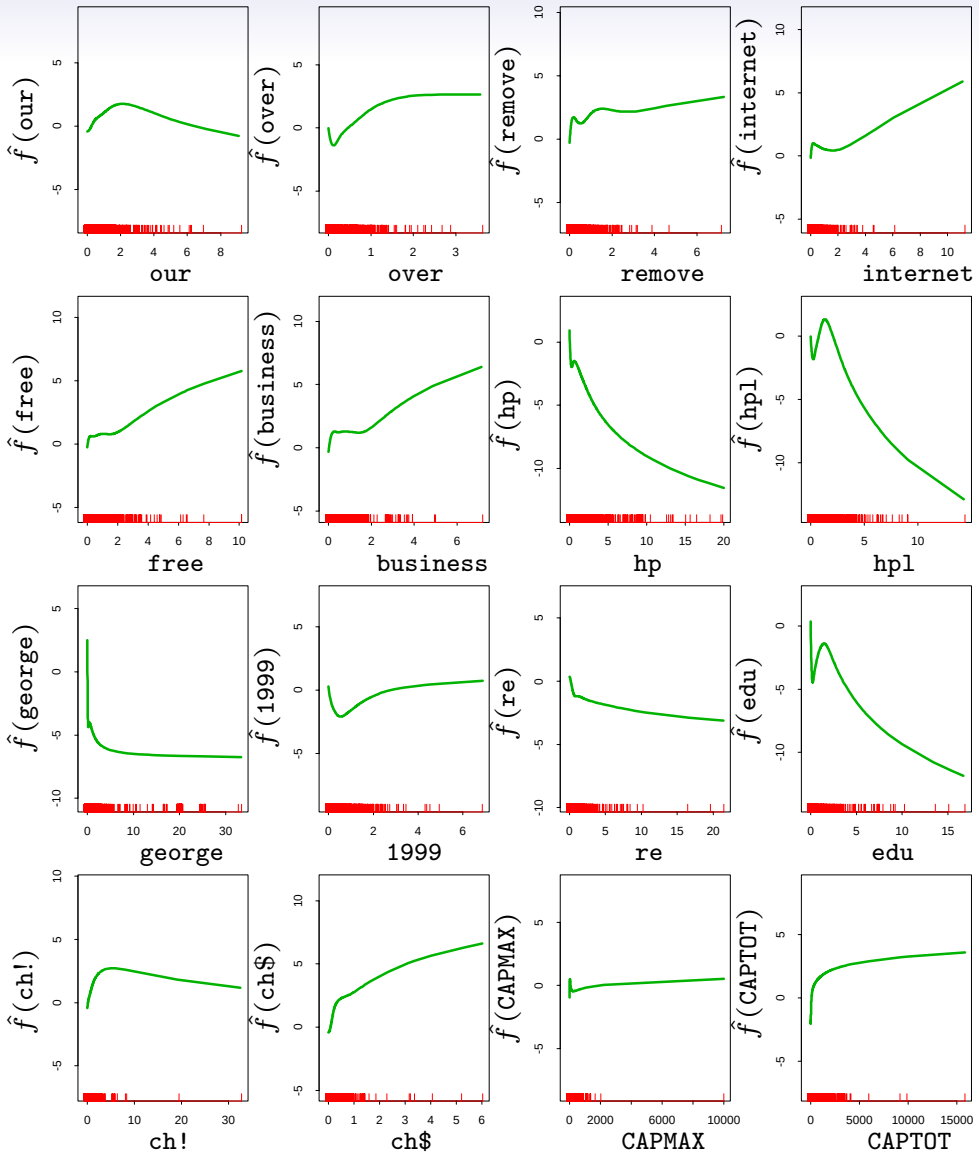
Fitting generalized additive models

Logistic Regression

- Compute starting values: f_j^{old} and $\eta^{old} = \sum_j f_j^{old}(x_j)$ e.g. using linear logistic regression
- Iterate
  - construct adjusted dependent variable

$$z_i = \eta_i^{old} + \frac{(y_i - p_i^{old})}{p_i^{old}(1 - p_i^{old})}$$

  - construct weights $w_i = p_i^{old}(1 - p_i^{old})$
  - compute $\eta^{new} = A_w z$, the weighted additive model fit to z.
- Stop when functions don't change



GAM fit to SPAM data

- Shown are the most important predictors.
- Many show nonlinear behavior.
- Overall error rate 5.3% .
- Functions can be re-parametrized (e.g. log terms, quadratic, step-functions), and then fit by linear model.
- Produces a prediction per email $\Pr(\text{SPAM}|X = x)$

Model selection with GAMs

- Gams have a tuning parameter for each term/variable in the model, to determine how smooth/wiggly each function should be. For example the λ_j in smoothing splines.
- In original GAM work, we proposed a kind of manual stepwise model fitting, using approximate ANOVA type tests.
- Does not scale to large number of variables.
- Automatic methods are more desirable.

`step.gam()` in package `gam`

- For each of p variables to be considered, provide an *ordered* list of alternatives:
  `scope[[j]] = ~ 1 + free + s(free,df=4) + s(free,df=8)`
- Provide a starting model:
  `start = gam(spam ~ 1, family = binomial, data = spam.train)`
- `step.gam` minimizes an AIC criterion by greedy ordered search:
  `step.gam(start, scope, direction = "both", ...)`

`step.gam()` in package `gam`

- For each of p variables to be considered, provide an *ordered* list of alternatives:
  `scope[[j]] = ~ 1 + free + s(free,df=4) + s(free,df=8)`
- Provide a starting model:
  `start = gam(spam ~ 1, family = binomial, data = spam.train)`
- `step.gam` minimizes an AIC criterion by greedy ordered search:
  `step.gam(start, scope, direction = "both", ...)`
- If p is large, go and get some coffee!

`step.gam()` in package `gam`

- For each of p variables to be considered, provide an *ordered* list of alternatives:
  `scope[[j]] = ~ 1 + free + s(free,df=4) + s(free,df=8)`
- Provide a starting model:
  `start = gam(spam ~ 1, family = binomial, data = spam.train)`
- `step.gam` minimizes an AIC criterion by greedy ordered search:
  `step.gam(start, scope, direction = "both", ...)`
- If p is large, go and get some coffee!
  `parallel = TRUE` can help.

`step.gam()` in package `gam`

- For each of p variables to be considered, provide an *ordered* list of alternatives:
  `scope[[j]] = ~ 1 + free + s(free,df=4) + s(free,df=8)`
- Provide a starting model:
  `start = gam(spam ~ 1, family = binomial, data = spam.train)`
- `step.gam` minimizes an AIC criterion by greedy ordered search:
  `step.gam(start, scope, direction = "both", ...)`
- If p is large, go and get some coffee!
  `parallel = TRUE` can help.
- If p is very large (in the 100s), go out for lunch!

BRUTO

Name means nothing!

BRUTO

Name means nothing! Not quite. Makes fun of Friedman and Silverman's TURBO — an anagram suggesting brute-force.

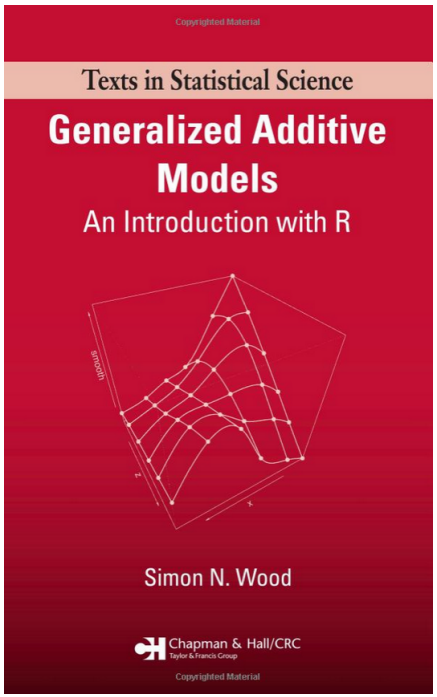
BRUTO

Name means nothing! Not quite. Makes fun of Friedman and Silverman's TURBO — an anagram suggesting brute-force.

$$\mathrm{GCV}^B(\lambda_1,\ldots,\lambda_p) = \frac{\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{p}\hat{f}_{j,\lambda_j}(x_{ij})\right)^2}{\left(1-[1+\sum_{j=1}^{p}\mathrm{df}_j]/N\right)^2}$$

- Optimize each λ_j separately in an adaptive type of backfitting; $\mathrm{df}_j = \mathrm{tr}(S_j(\lambda_j))$. Optimization of λ_j done by `smooth.spline()` in `R`.
- Apart from a continuous range of λ_j for jth variable, a *linear* term, and a *null* term are also considered.
- `bruto` is available in `mda` package in R, for squared-error loss. In principal, could be extended to other loss functions, but has never been as far as I know.

GAM.SELECTION in package MGCV





- This terrific 2006 book saved us writing a second edition to our GAM book. `gam{mgcv}` is a popular implementation of GAMS.
- `gam.selection` focuses on using GCV to select multiple smoothing parameters, although recent changes also allow for variable selection, similar to BRUTO.

Selection using Convex Optimization

- In 1996, Rob Tibshirani introduces the LASSO, a convex approach to variable selection and regularization:

$$\text{minimize } L(y, X\beta) + \lambda\|\beta\|_1,$$

  where $\|\beta\|_1 = \sum_j |\beta_j|$. For λ large, solutions are sparse.
- In 2006, Yuan and Lin generalize to the *group lasso*, for selecting coefficient vectors for groups of variables:

$$\text{minimize } L(y, \sum_k X_k\theta_k) + \lambda \sum_k \|\theta_k\|_2.$$

- By varying λ, we trace out a *solution path* with varying sparsity.

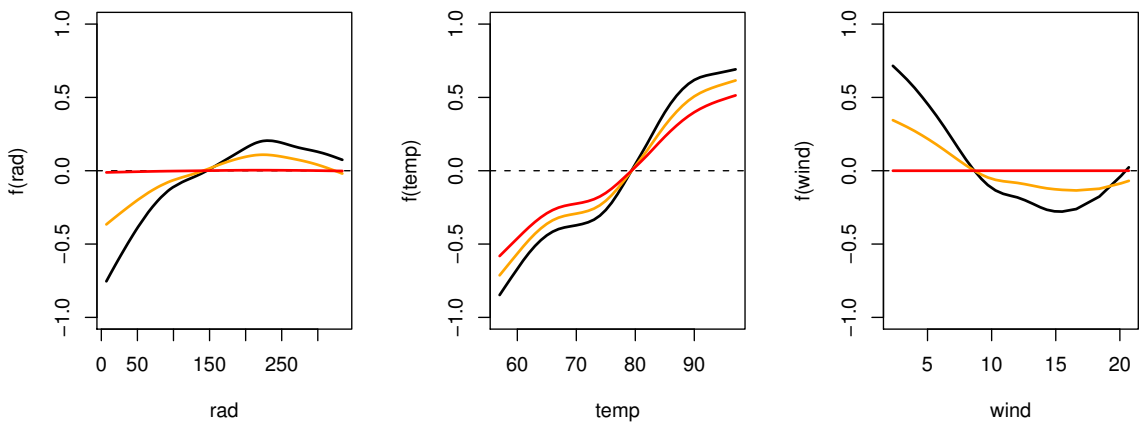
SPAM: Sparse Additive Models

Ravikumar, Liu, Lafferty and Wasserman (2009)

$$\underset{\{f_j\}_1^p}{\text{minimize}}\, E(Y - \sum_{j=1}^{p} f_j(X_j))^2 + \lambda \sum_{j=1}^{p} \|f_j\|_2$$

They implement this with smoothers and a modified type of backfitting:

$$\widetilde{f}_j = S_j\big(y - \sum_{k \neq j} \widehat{f}_k\big) \quad \text{and} \quad \widehat{f}_j = \left(1 - \frac{\lambda}{\|\widetilde{f}\|_2}\right)_+ \widetilde{f}_j,$$



COSSO: Component Selection and Smoothing Operator

Lin and Zhang (2003).
Smoothing-spline additive model

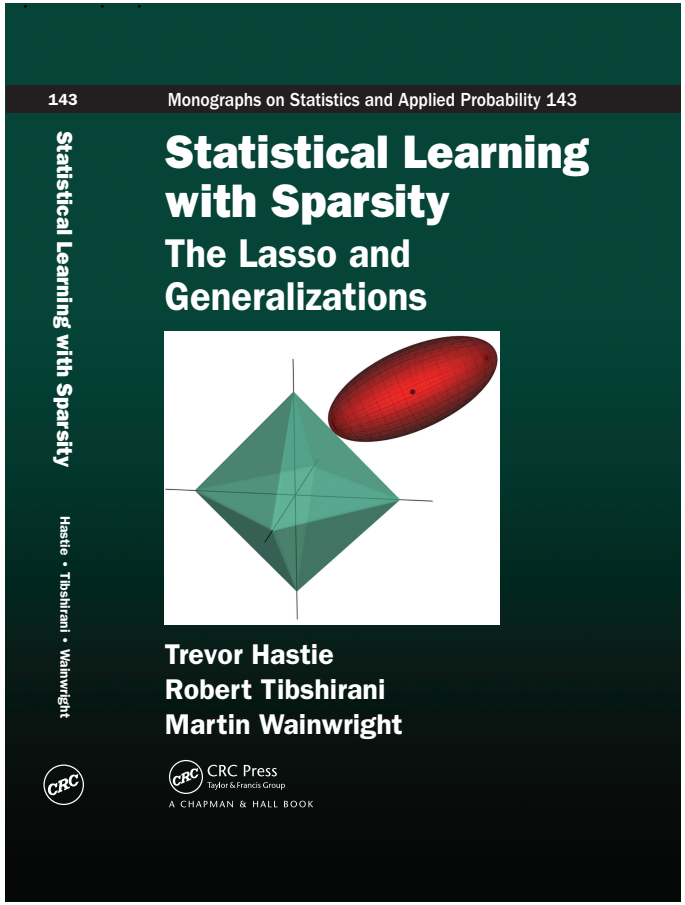
$$\underset{\{f_j\}_1^p}{\text{minimize}} \sum_{i=1}^{N} (y_i - \sum_j f_j(x_{ij}))^2 + \lambda \sum_j \int f_j''(t)^2 dt$$

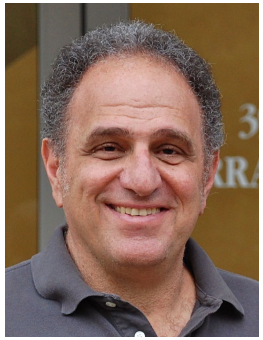
has finite-dimensional RKHS solution

$$\underset{\{\theta_j\}_1^p}{\text{minimize}} \|y - \sum_j R_j \theta_j\|)_2^2 + \lambda \sum_j \|\theta_j\|_{R_j}^2$$

where $\|\theta_j\|_{R_j}^2 = \theta_j^T R_j \theta_j$. COSSO solves instead

$$\underset{\{\theta_j\}_1^p}{\text{minimize}} \|y - \sum_j R_j \theta_j\|)_2^2 + \lambda \sum_j \|\theta_j\|_{R_j}$$







Published May 2015

Includes discussion of group lasso, SPAM and COSSO, as well as lasso and generalizations to wide variety of statistical problems.

GAMSEL: Generalized Additive Model Selection



Joint work with recent Ph.D graduate Alexandra Chouldechova. Paper submitted to journal and arXiv in June 2015.

Uses *overlap group lasso* to perform automatic, *sticky* selection between zero, linear or nonlinear terms in GAMs:

$$\eta(x) = \sum_{j=1}^{p} f_j(x_j)$$

Package `gamsel` fits regularization path for gaussian and binomial generalized additive models.

GAMSEL optimization criterion

$$\frac{1}{2}\left\| y - \sum_{j=1}^{p} \alpha_j x_j - \sum_{j=1}^{p} U_j \beta_j \right\|^2 + \lambda \sum_{j=1}^{p} \left\{ (1-\gamma)|\alpha_j| + \gamma \|\beta_j\|_{D_j^*} \right\} + \frac{1}{2} \sum_{j=1}^{p} \psi_j \|\beta_j\|_{D_j}^2$$

- $U_j = [x_j \; p_1(x_j) \; \cdots \; p_k(x_j)]$ where the p_i are orthogonal Demmler-Reinsch spline basis functions of increasing degree.
- $D_j = \text{diag}(d_{j0}, d_{j1}, \ldots, d_{jk})$ diagonal penalty matrix with $0 = d_{j0} < d_{j1} \leq d_{j2} \leq \cdots \leq d_{jk}$, and $D_j^* = D_j$ but with $d_{j0} = d_{j1}$.

$$\frac{1}{2}\left\|y-\sum_{j=1}^{p}\alpha_j x_j-\sum_{j=1}^{p}U_j\beta_j\right\|^2 \quad + \quad \lambda\sum_{j=1}^{p}\left\{(1-\gamma)|\alpha_j|+\gamma\|\beta_j\|_{D_j^*}\right\}$$

$$+ \quad \frac{1}{2}\sum_{j=1}^{p}\psi_j\|\beta_j\|_{D_j}^2$$

- When $\lambda = 0$ solution is additive spline model, with *end-path* df determined by the ψ_j.

$$\frac{1}{2}\left\|y-\sum_{j=1}^{p}\alpha_j x_j-\sum_{j=1}^{p}U_j\beta_j\right\|^2 \quad + \quad \lambda\sum_{j=1}^{p}\left\{(1-\gamma)|\alpha_j|+\gamma\|\beta_j\|_{D_j^*}\right\}$$

$$+ \quad \frac{1}{2}\sum_{j=1}^{p}\psi_j\|\beta_j\|_{D_j}^2$$

- When $\lambda = 0$ solution is additive spline model, with *end-path* df determined by the ψ_j.
- When $\lambda > 0$
  - if $\beta_j = 0$ and $\alpha_j = 0$, the jth term is omitted;
  - if $\beta_j = 0$ and $\alpha_j \neq 0$, the jth term is linear;
  - if $\beta_j \neq 0$ both linear and nonlinear terms are included, irrespective of α_j.
- $\gamma \in [0, 1]$ weighs preference for linear versus nonlinear terms.

Some details

- The U_j have typically 10–12 columns, and are computed once upfront. We use *pseudosplines* (TH 1996) to approximate the Demmler-Reinsch basis. Could also use ideas from *P-splines* (Eilers amd Marx 1996) instead.

Some details

- The U_j have typically 10–12 columns, and are computed once upfront. We use *pseudosplines* (TH 1996) to approximate the Demmler-Reinsch basis. Could also use ideas from *P-splines* (Eilers amd Marx 1996) instead.
- We use blockwise coordinate descent to solve the objective, with each α_j (trivial) and β_j (less trivial) a block. We make heavy use of active-set methods and warm starts to compute the entire solution path.

Some details

- The U_j have typically 10–12 columns, and are computed once upfront. We use *pseudosplines* (TH 1996) to approximate the Demmler-Reinsch basis. Could also use ideas from *P-splines* (Eilers amd Marx 1996) instead.
- We use blockwise coordinate descent to solve the objective, with each α_j (trivial) and β_j (less trivial) a block. We make heavy use of active-set methods and warm starts to compute the entire solution path.
- We use *strong rules* for screening variables at each value of λ; these use heuristics to prune the active set, leading to dramatic savings in computational time.

Some details

- The U_j have typically 10–12 columns, and are computed once upfront. We use *pseudosplines* (TH 1996) to approximate the Demmler-Reinsch basis. Could also use ideas from *P-splines* (Eilers amd Marx 1996) instead.
- We use blockwise coordinate descent to solve the objective, with each α_j (trivial) and β_j (less trivial) a block. We make heavy use of active-set methods and warm starts to compute the entire solution path.
- We use *strong rules* for screening variables at each value of λ; these use heuristics to prune the active set, leading to dramatic savings in computational time.
- `gamsel()` is fast! 6s to compute the entire solution path [50 λs] with N=5000 and p=200.

Simulation

I could show you reams of results where `gamsel` outperforms its competitors, on (carefully chosen) examples. But I will spare you the pain — some comparisons are in the paper.

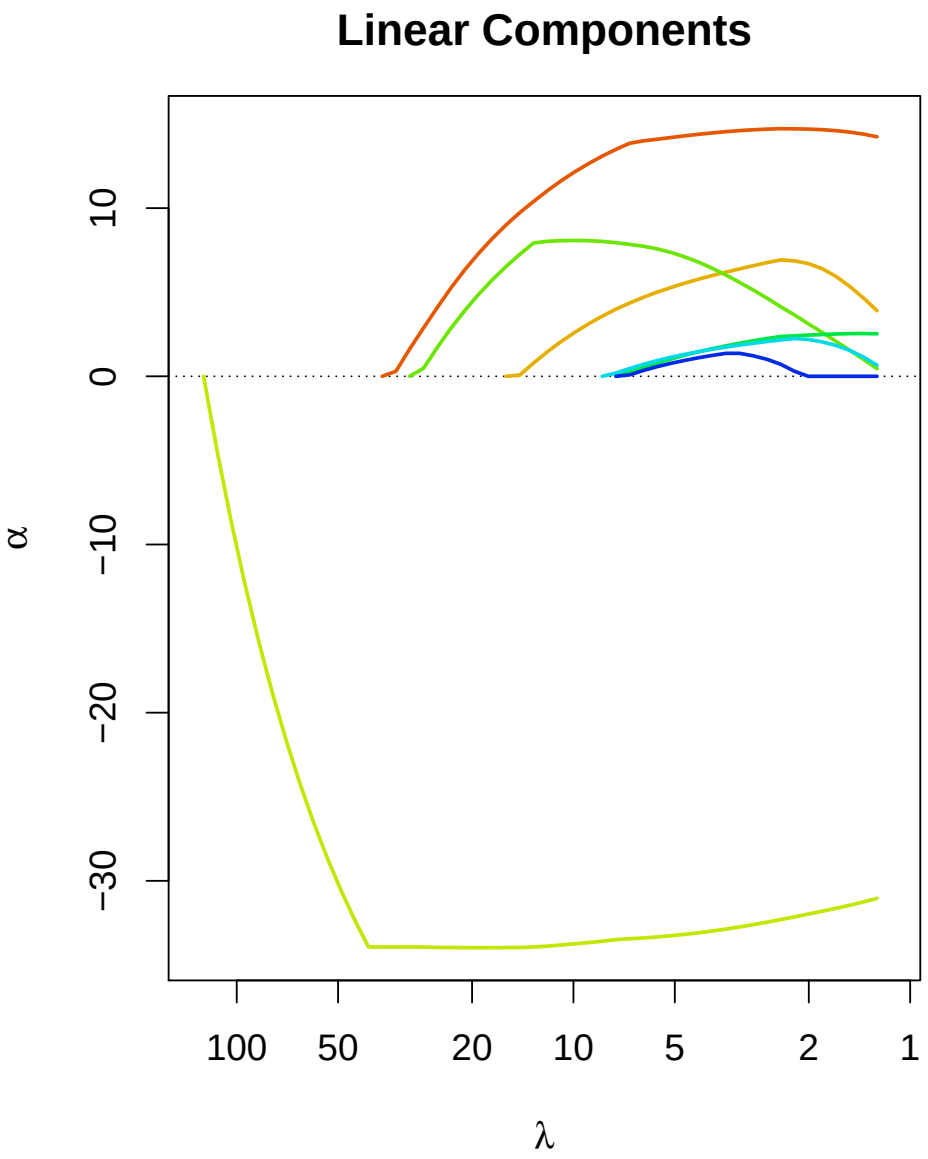
Simulation

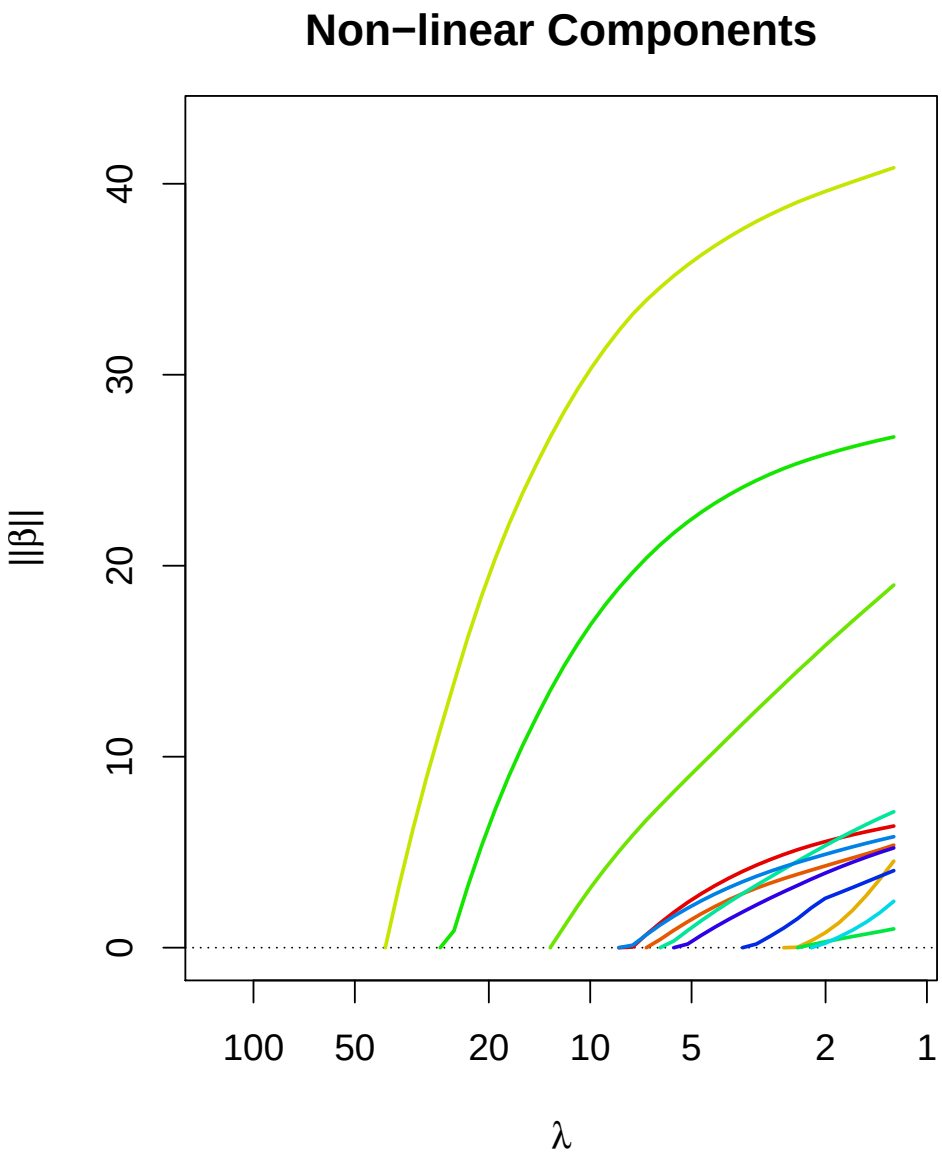
I could show you reams of results where `gamsel` outperforms its competitors, on (carefully chosen) examples. But I will spare you the pain — some comparisons are in the paper.

Instead I will show you what it does on a simulated example.

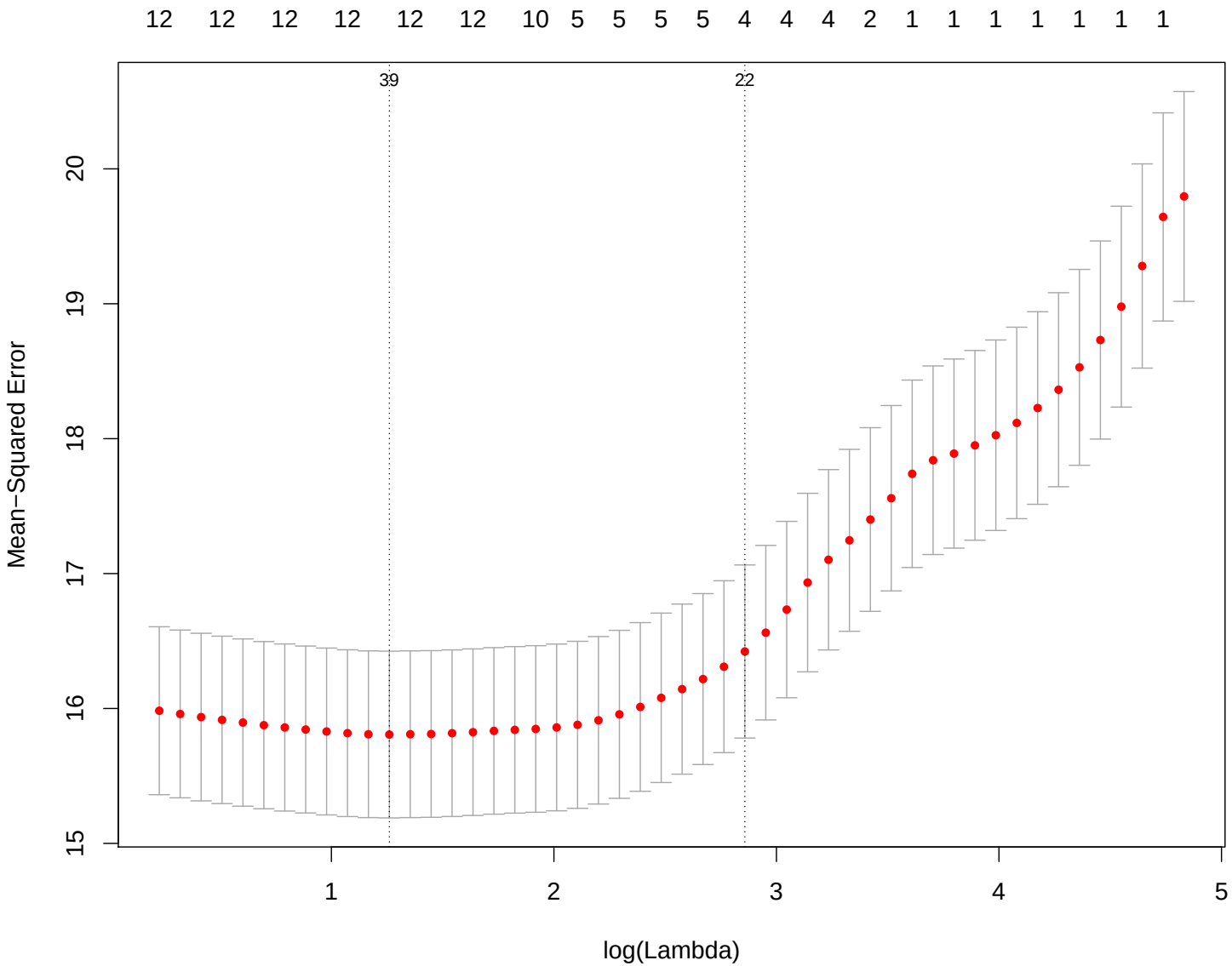
16 variables
Vars 1–3 linear
Vars 4–6 nonlinear
Vars 7–16 zero

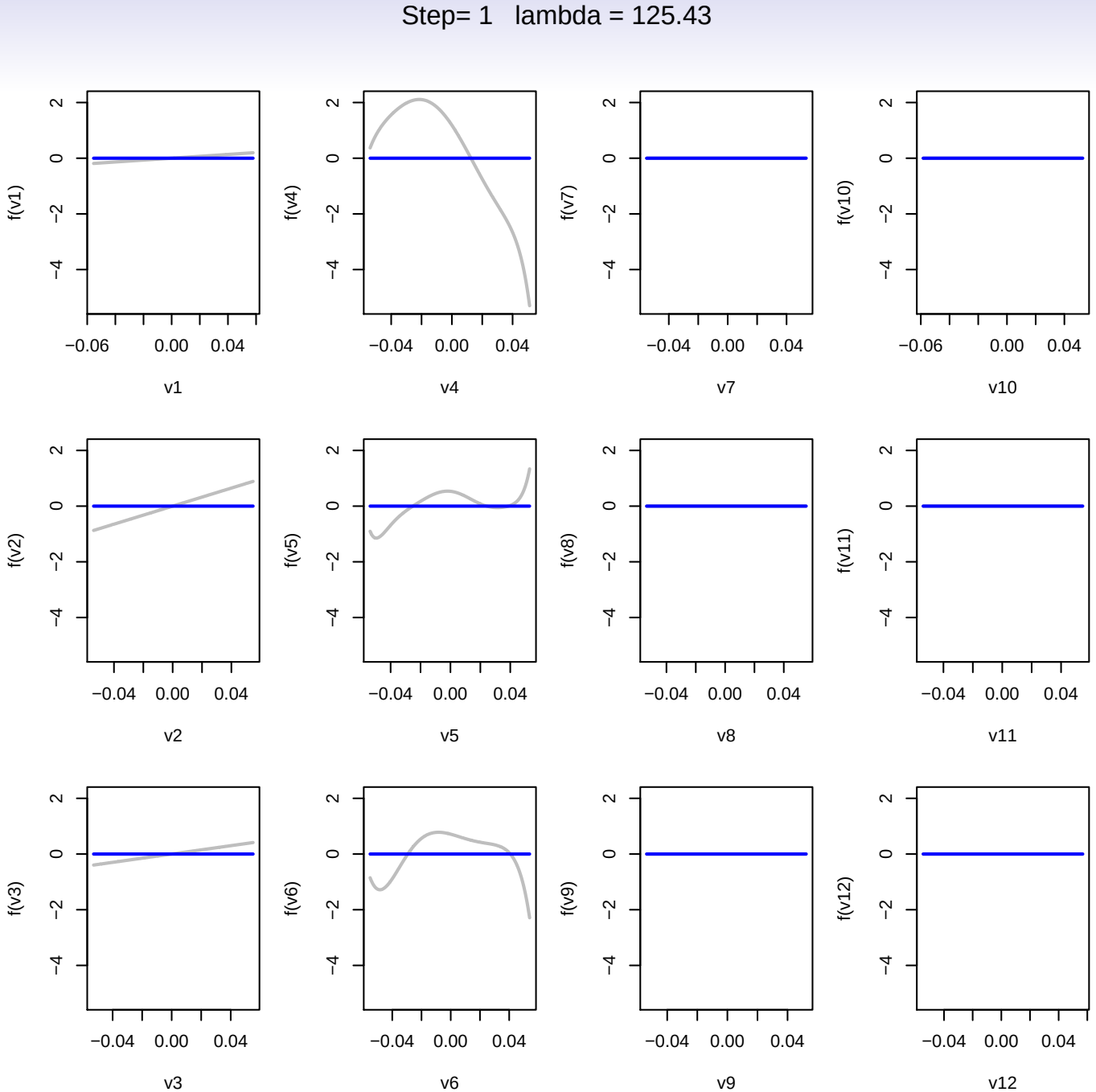
Summary of Path

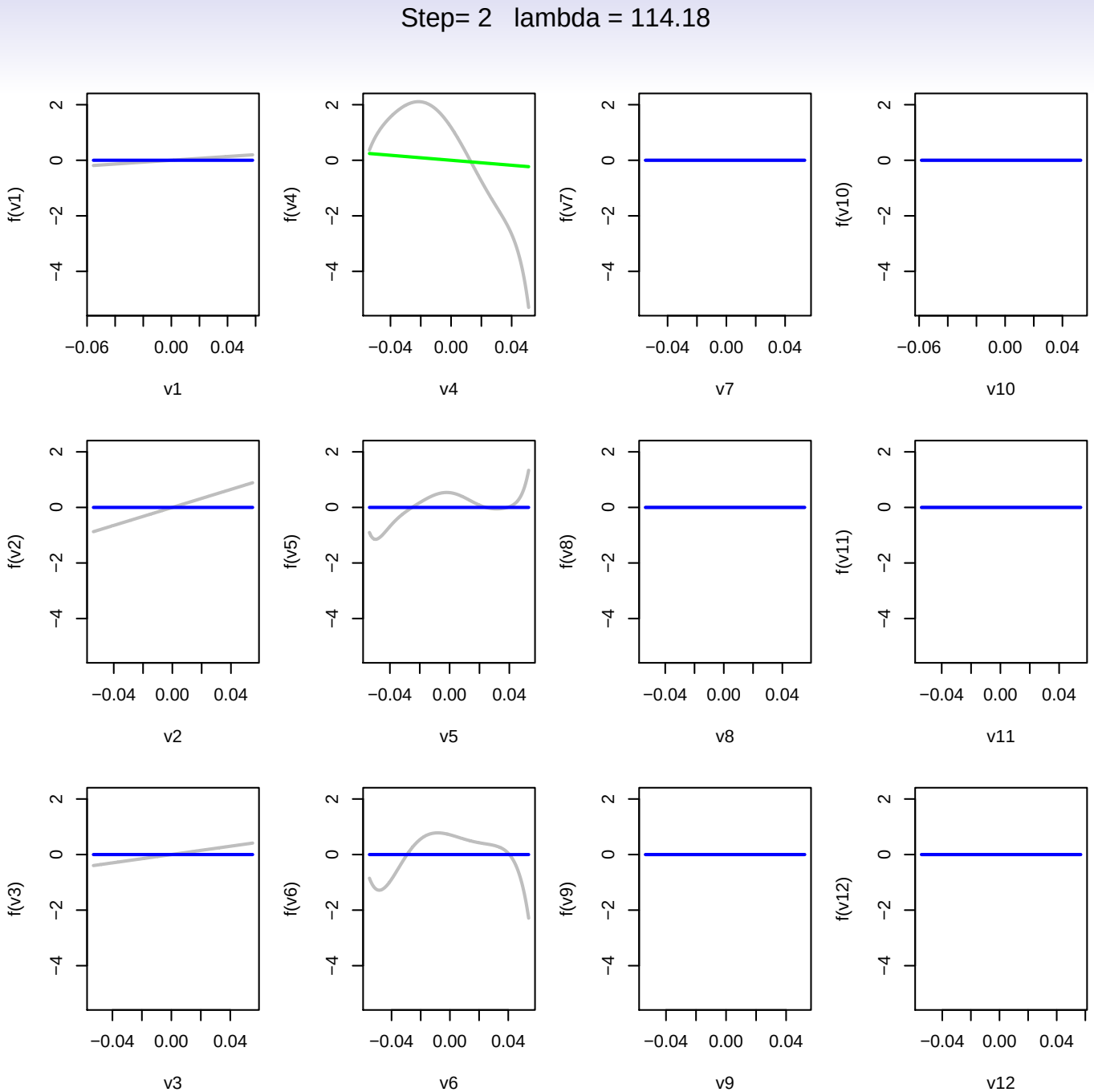


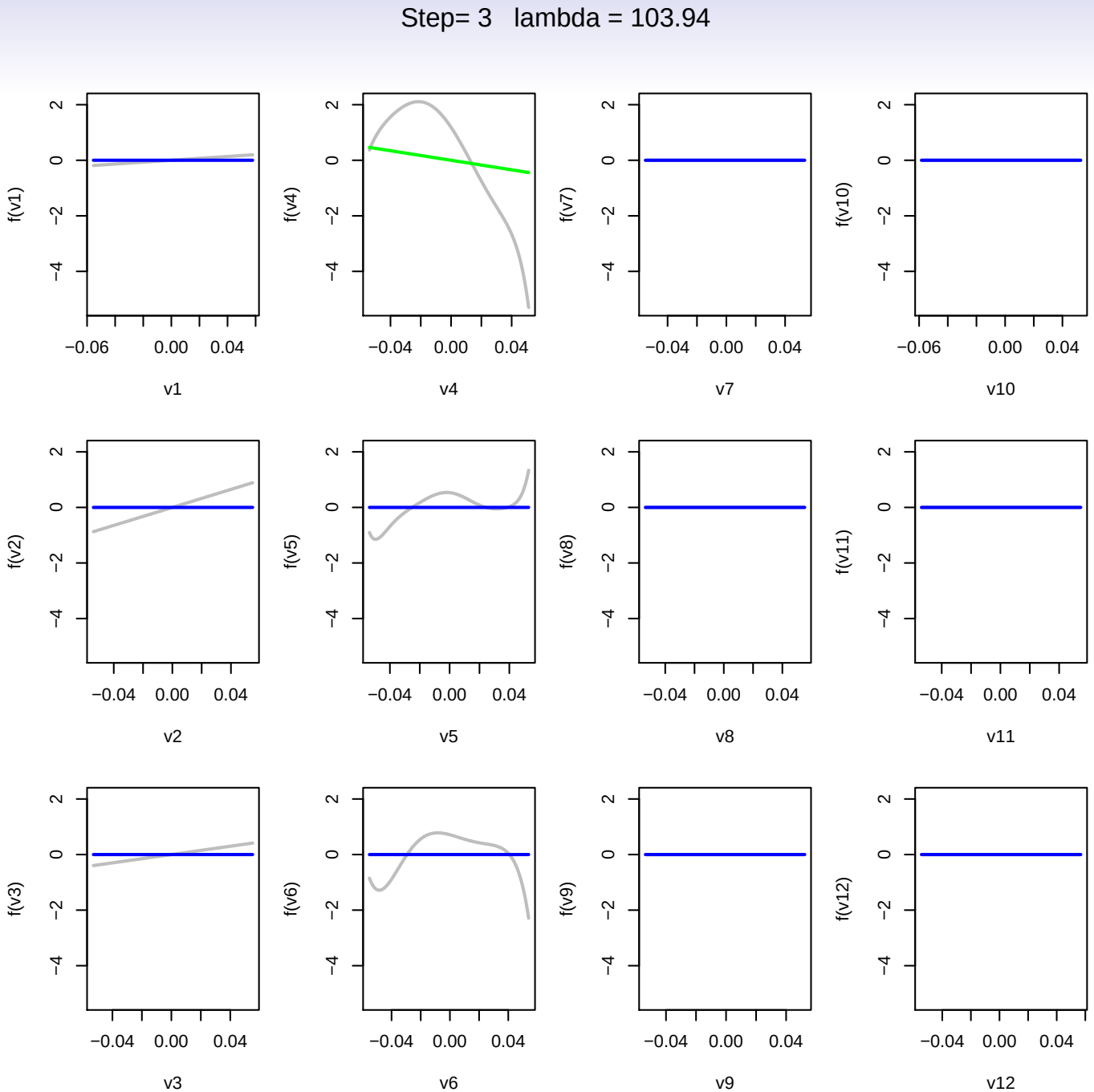


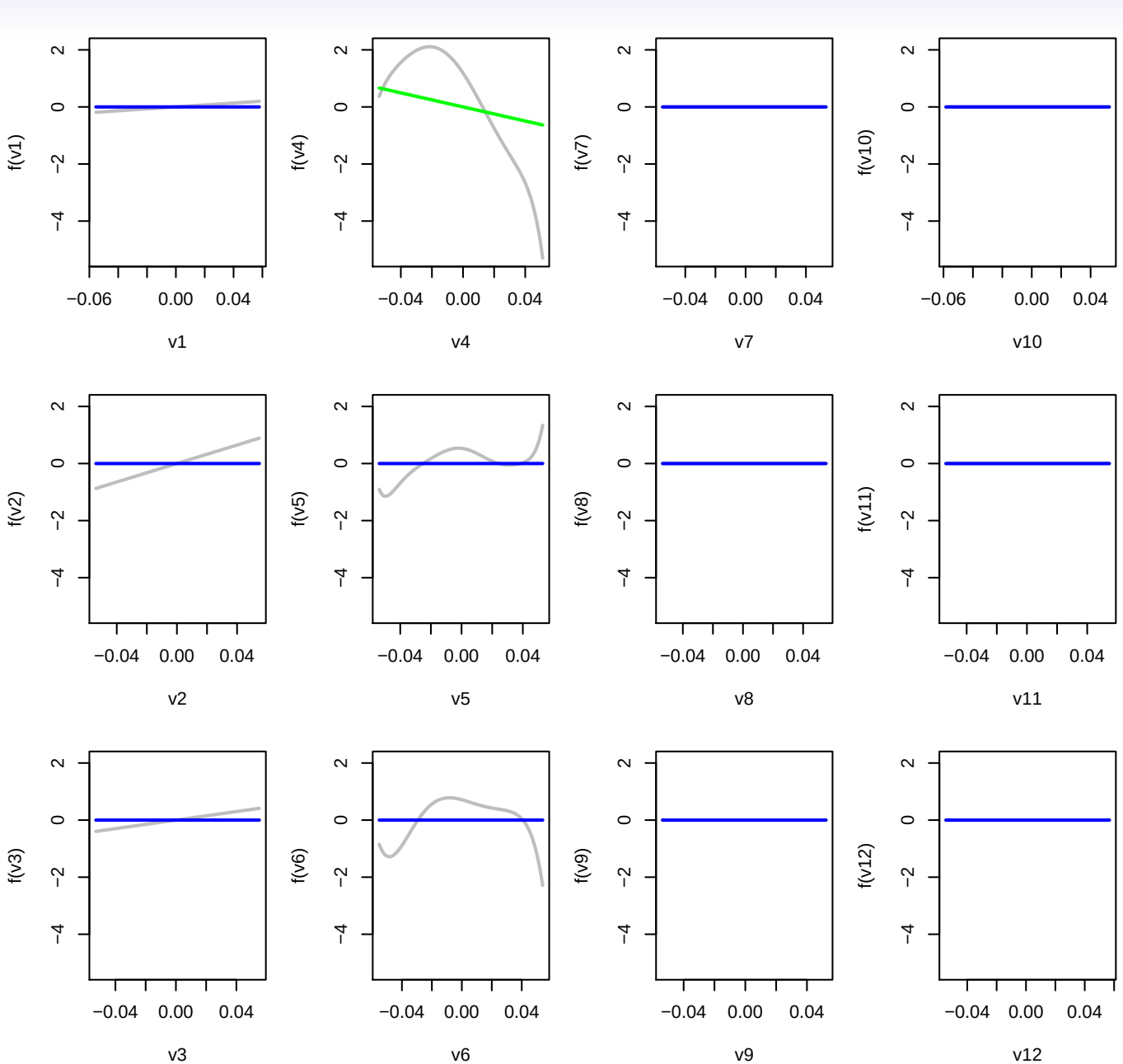
Cross-validation to pick λ



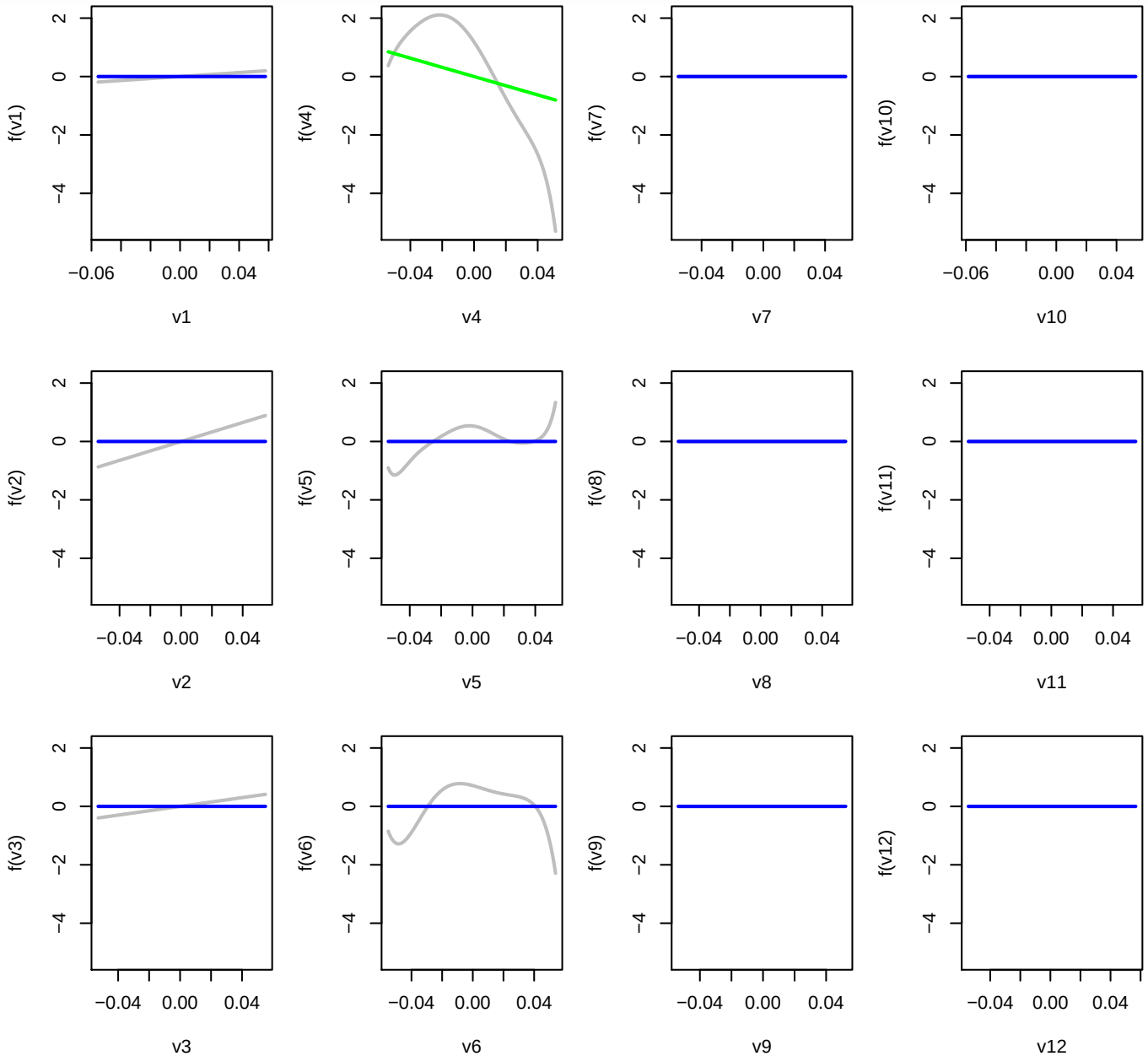

Step= 1 lambda = 125.43
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12

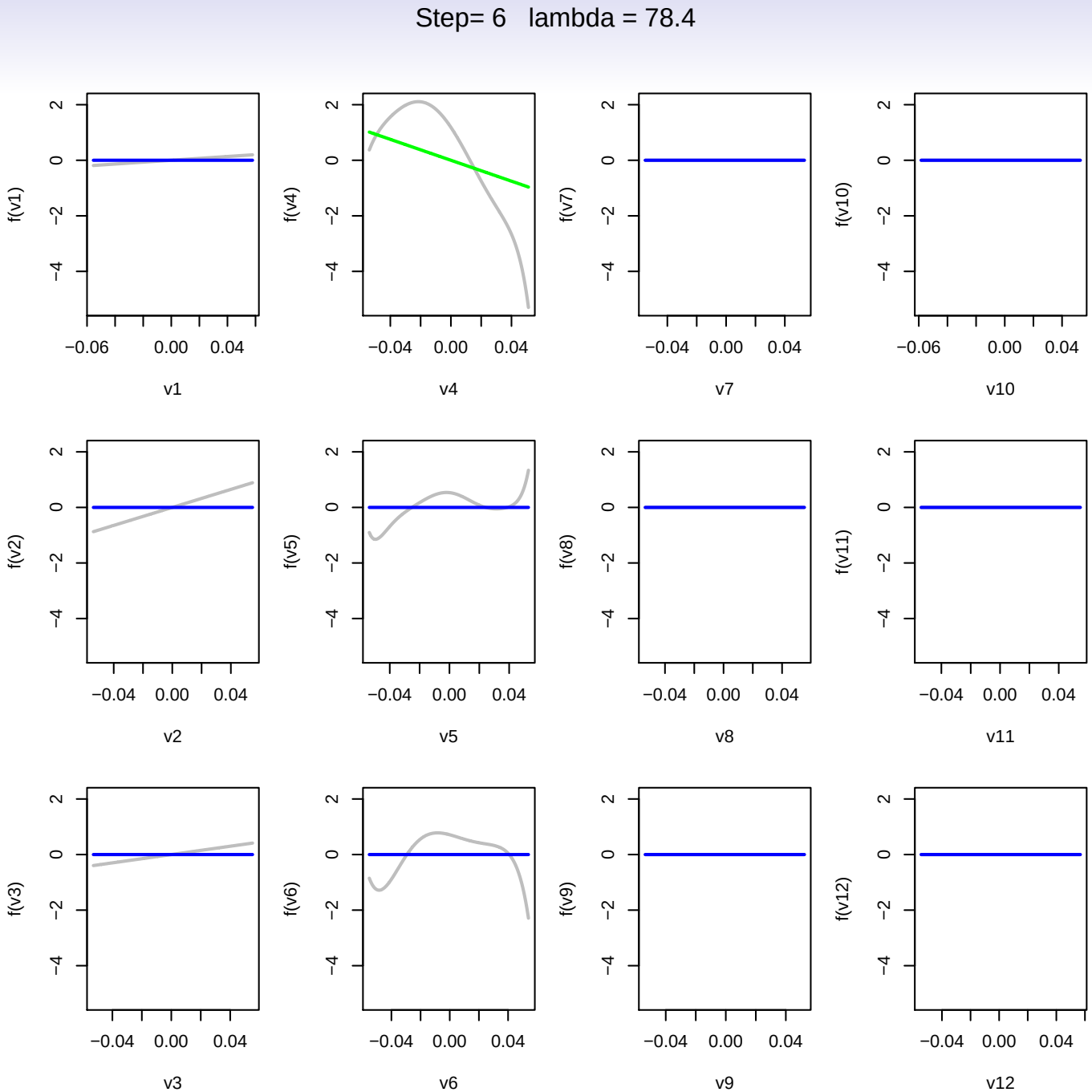

Step= 2 lambda = 114.18
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12


Step= 3 lambda = 103.94
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

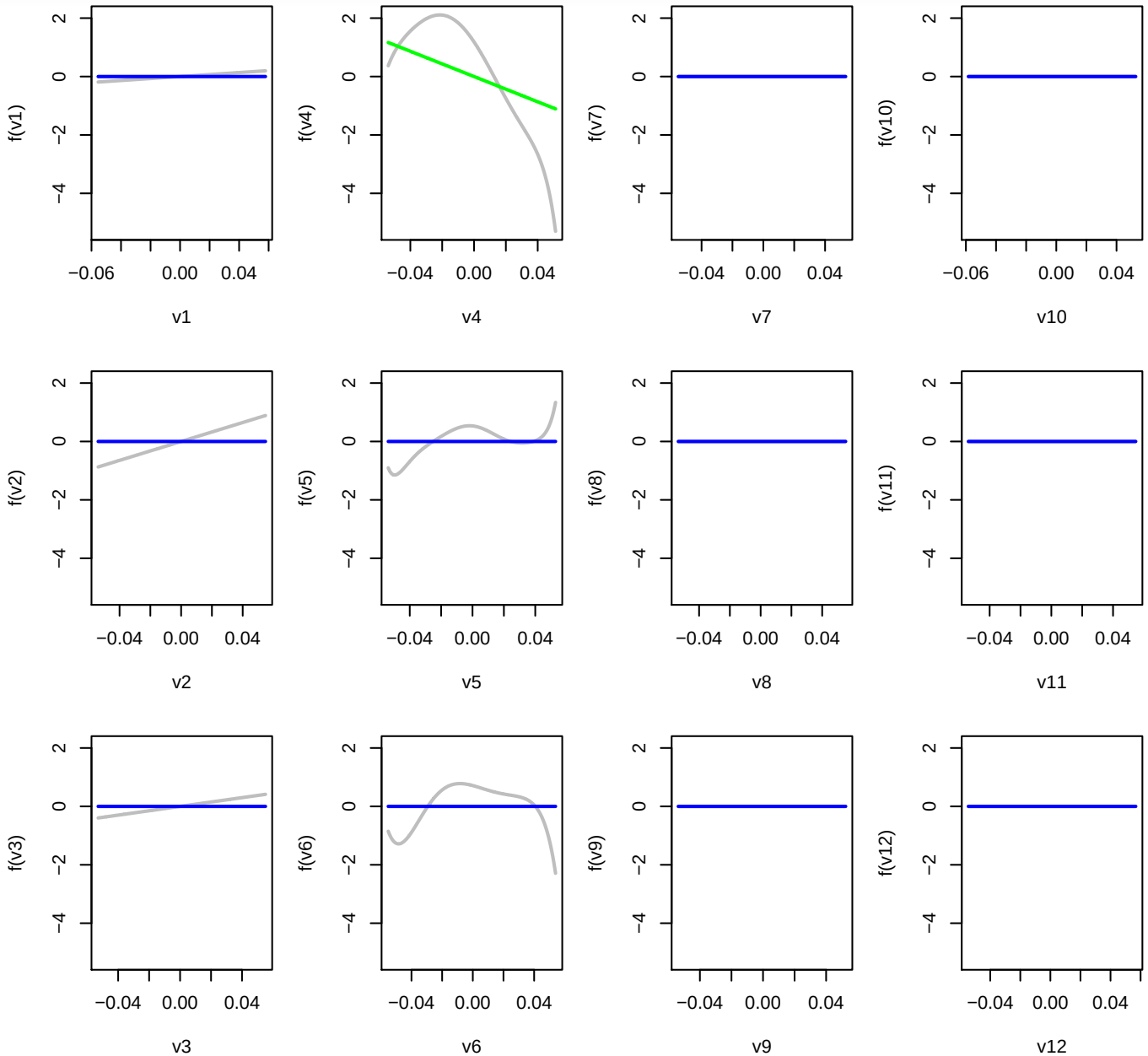

Step= 4 lambda = 94.61
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12

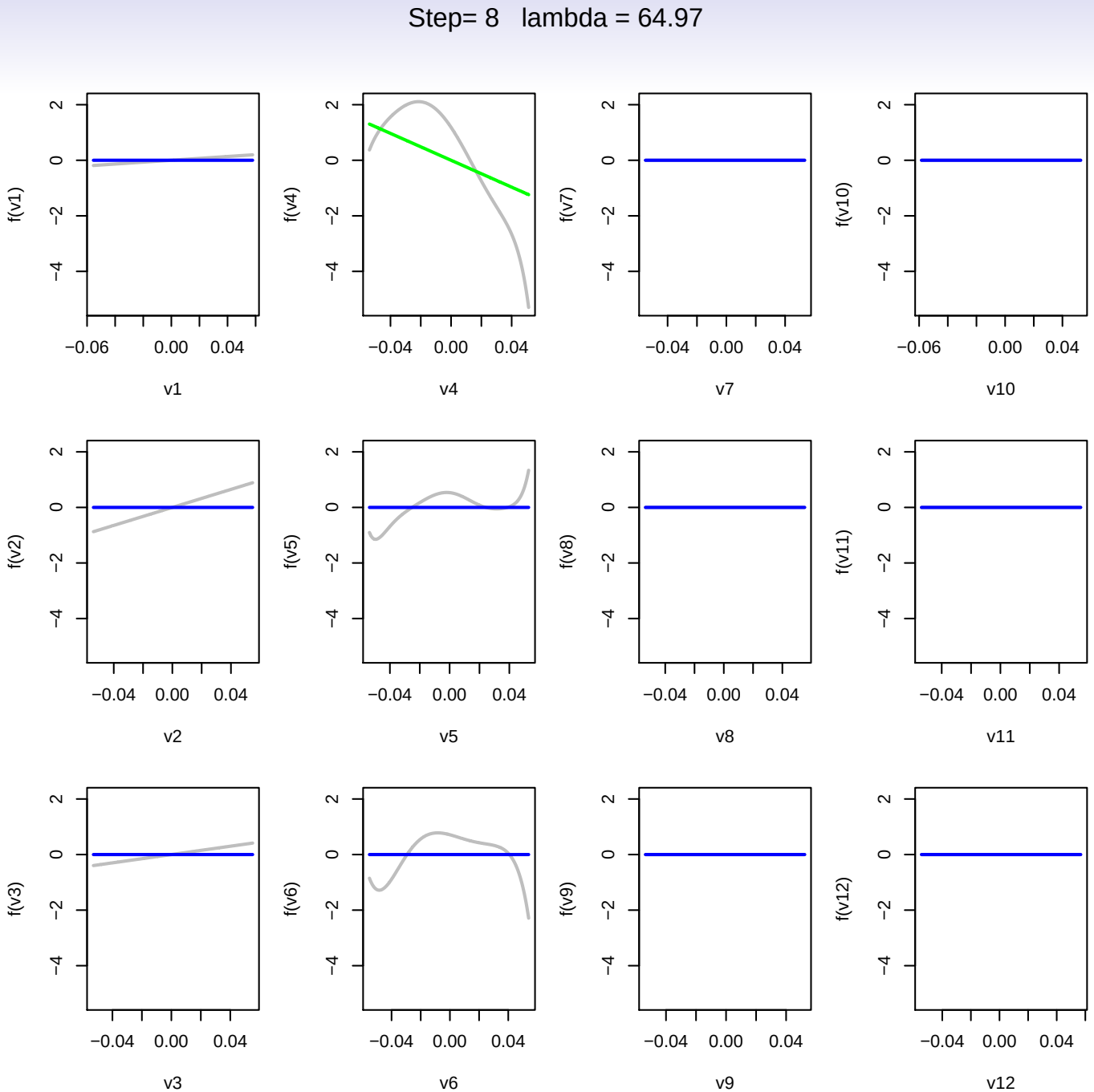
Step= 5 lambda = 86.13

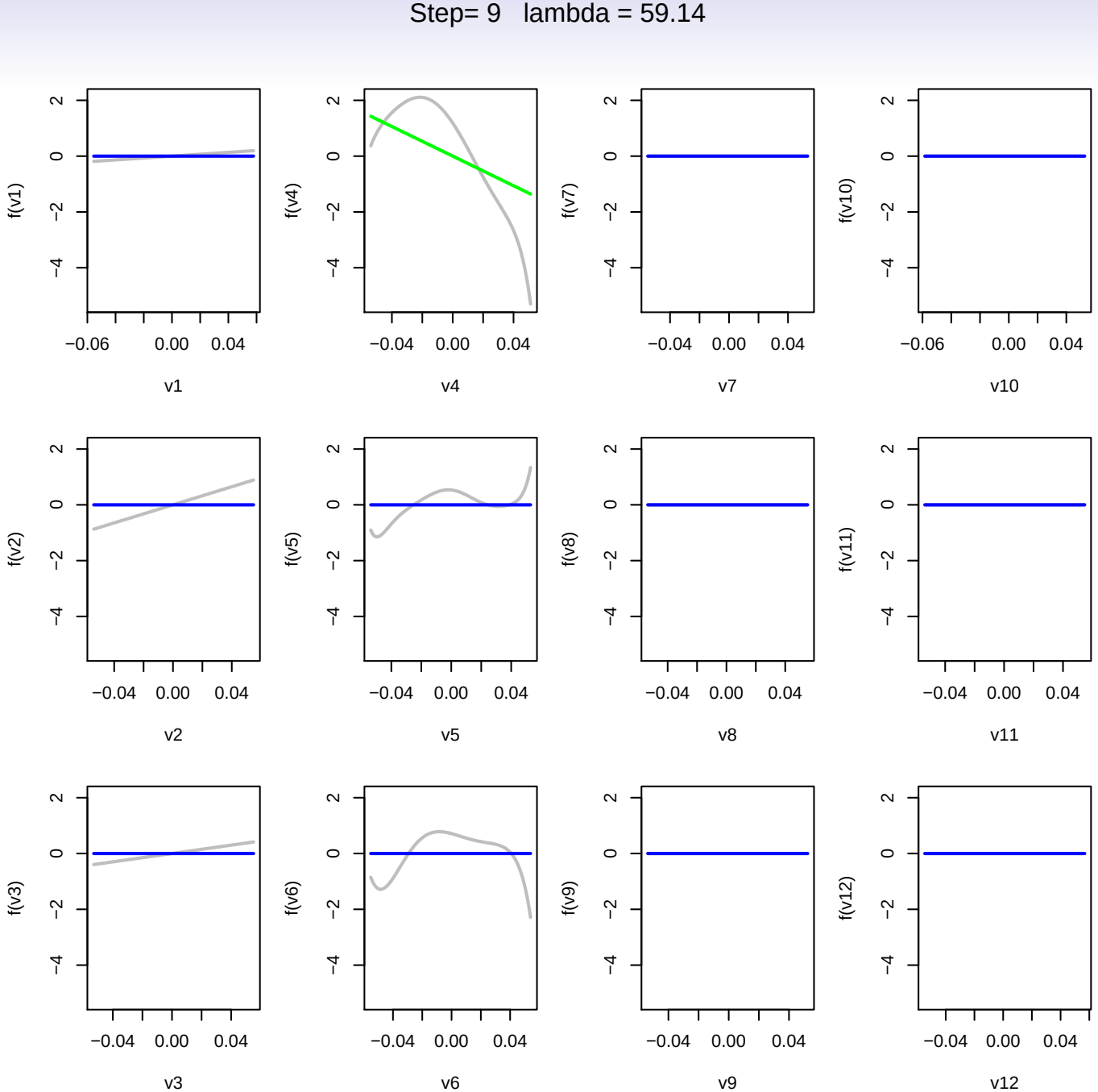



Step= 6 lambda = 78.4
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12

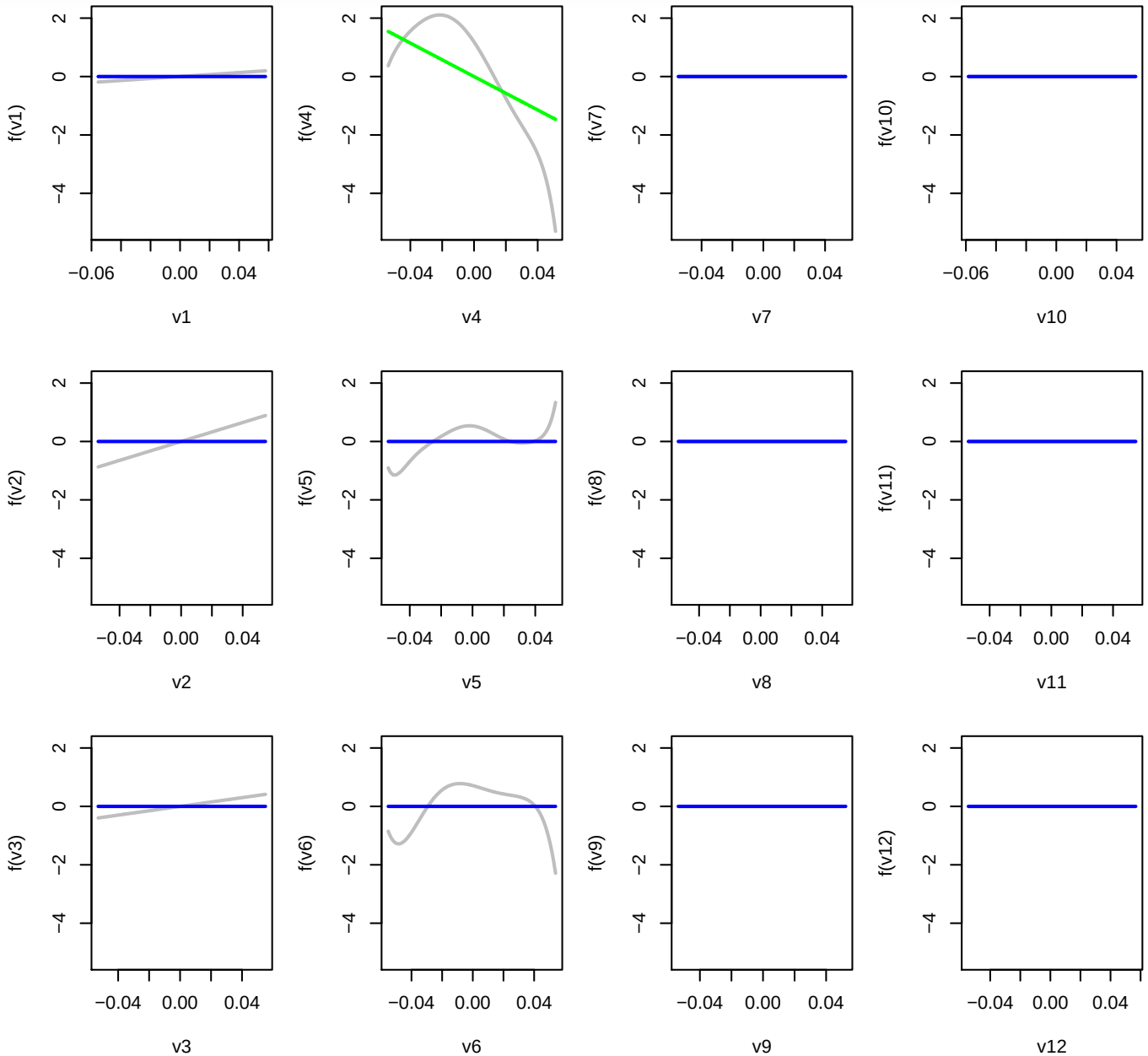
Step= 7 lambda = 71.37

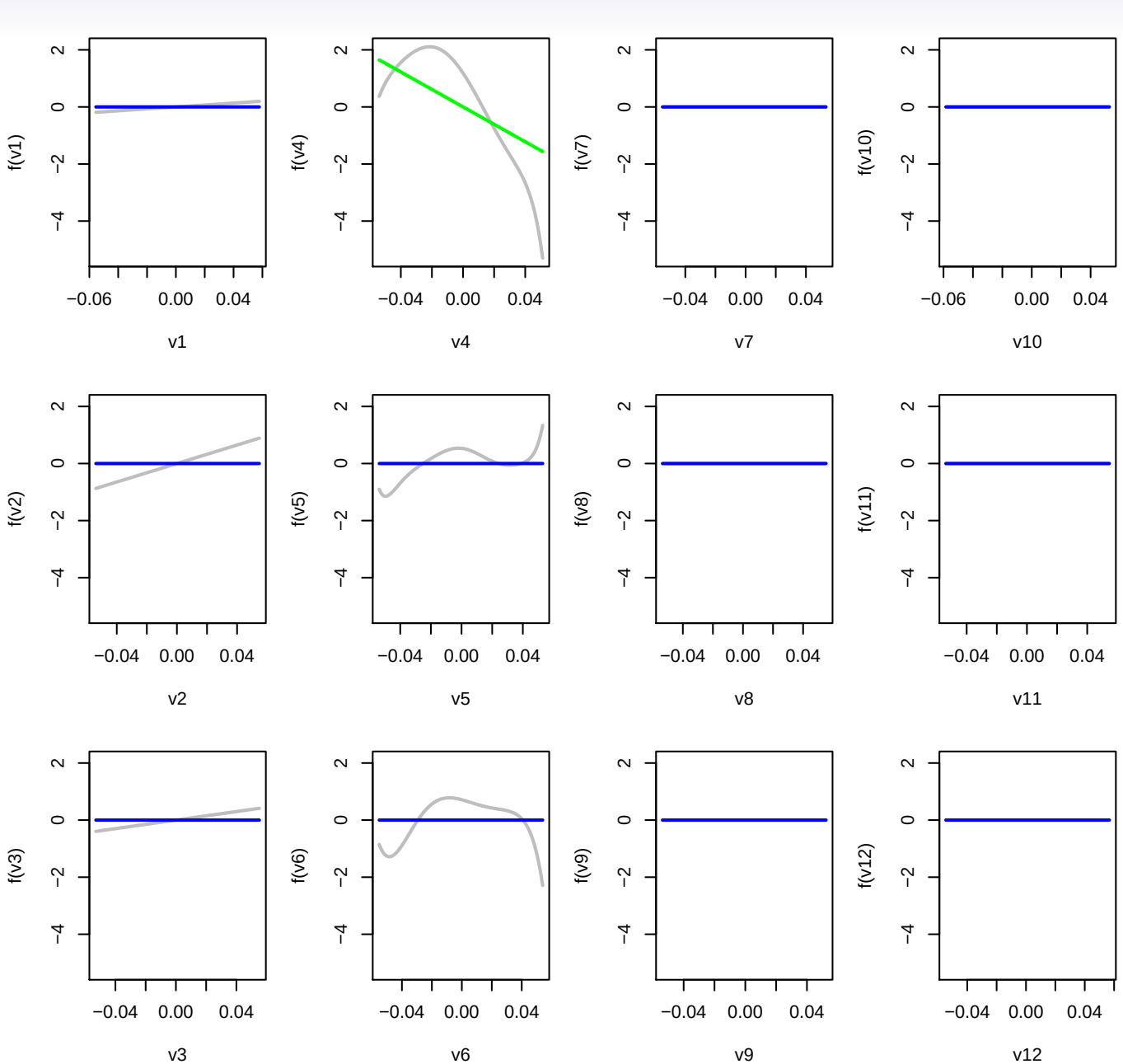


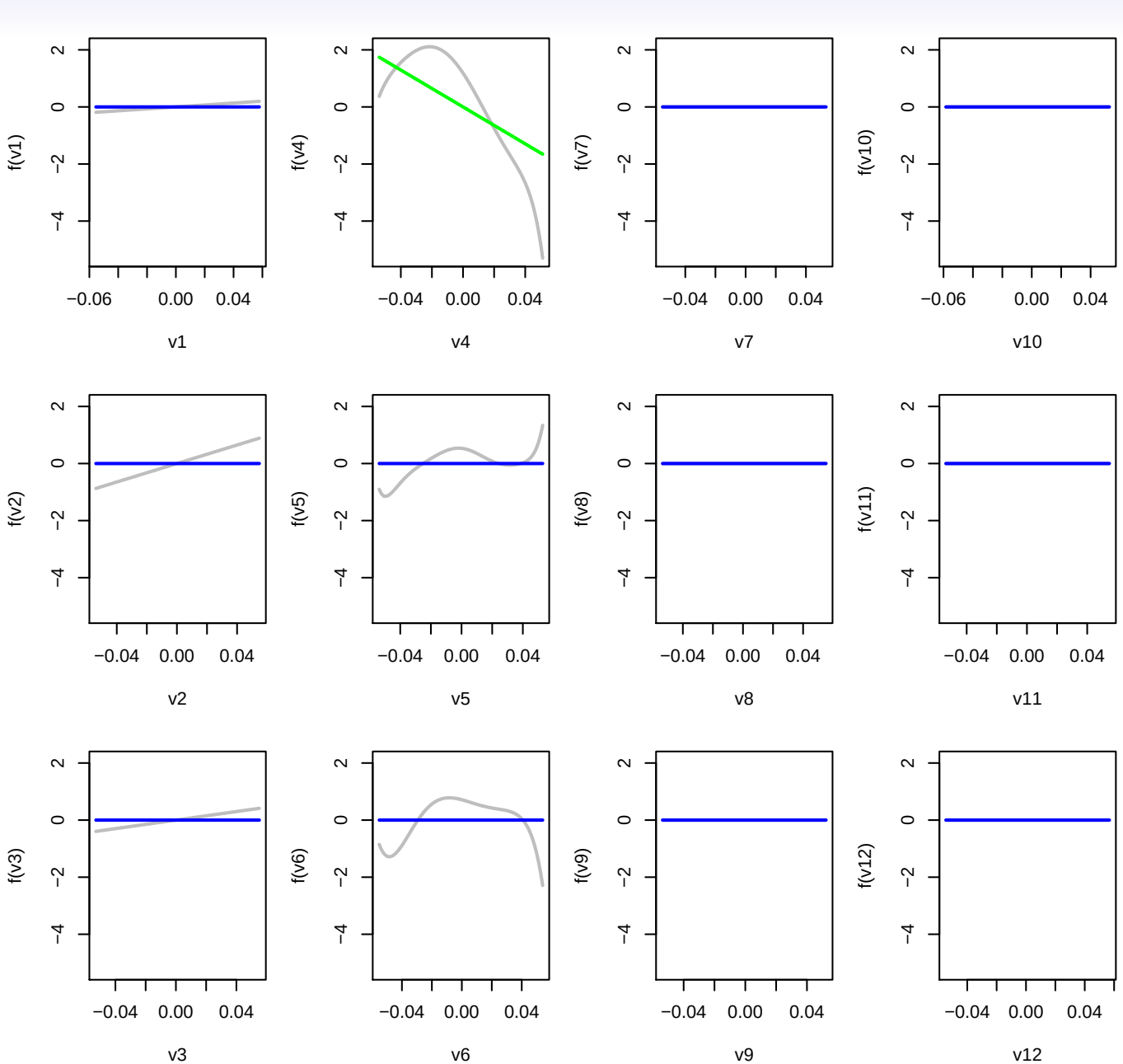

Step= 8 lambda = 64.97
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12


Step= 9 lambda = 59.14
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

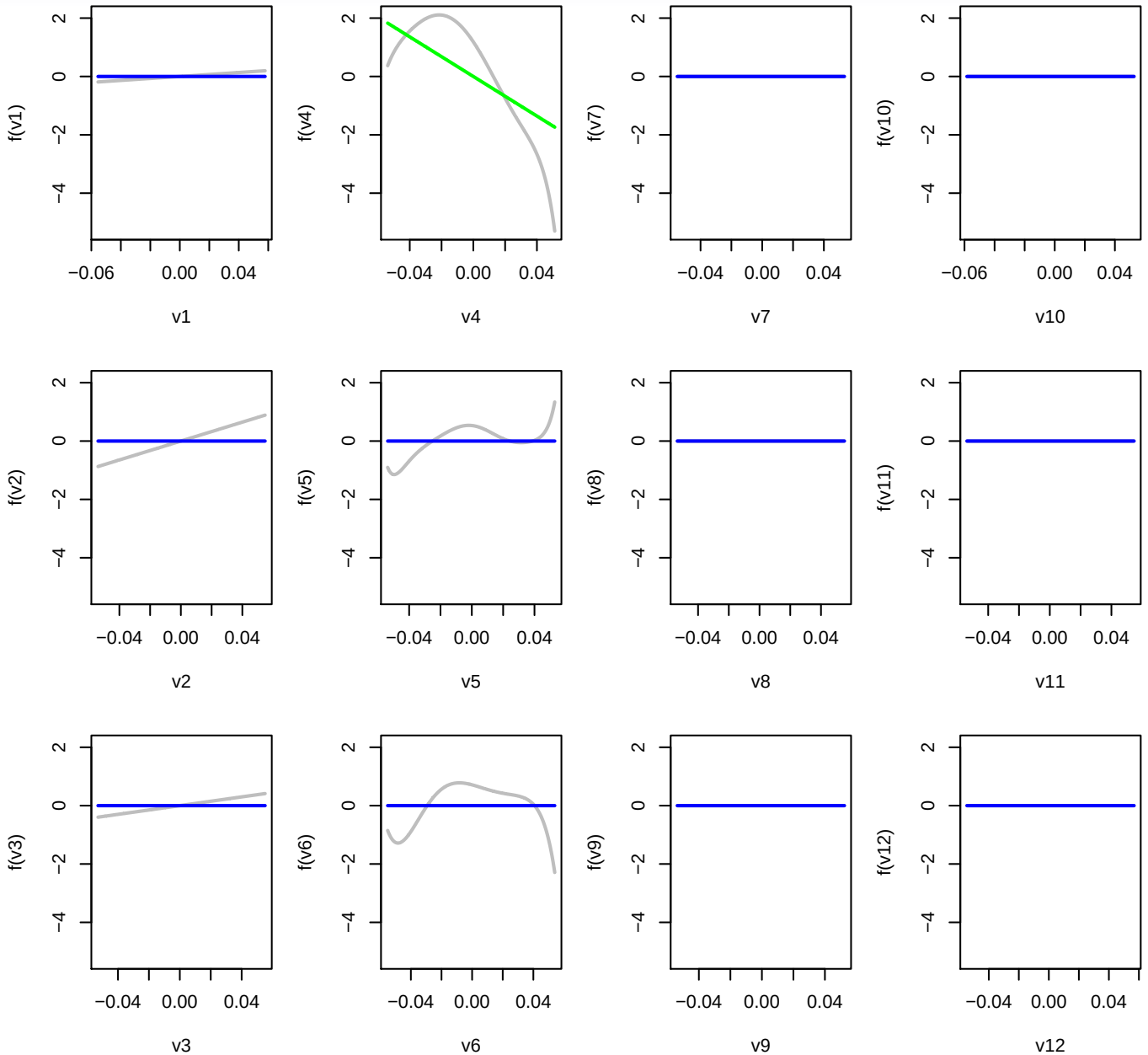
Step= 10 lambda = 53.83



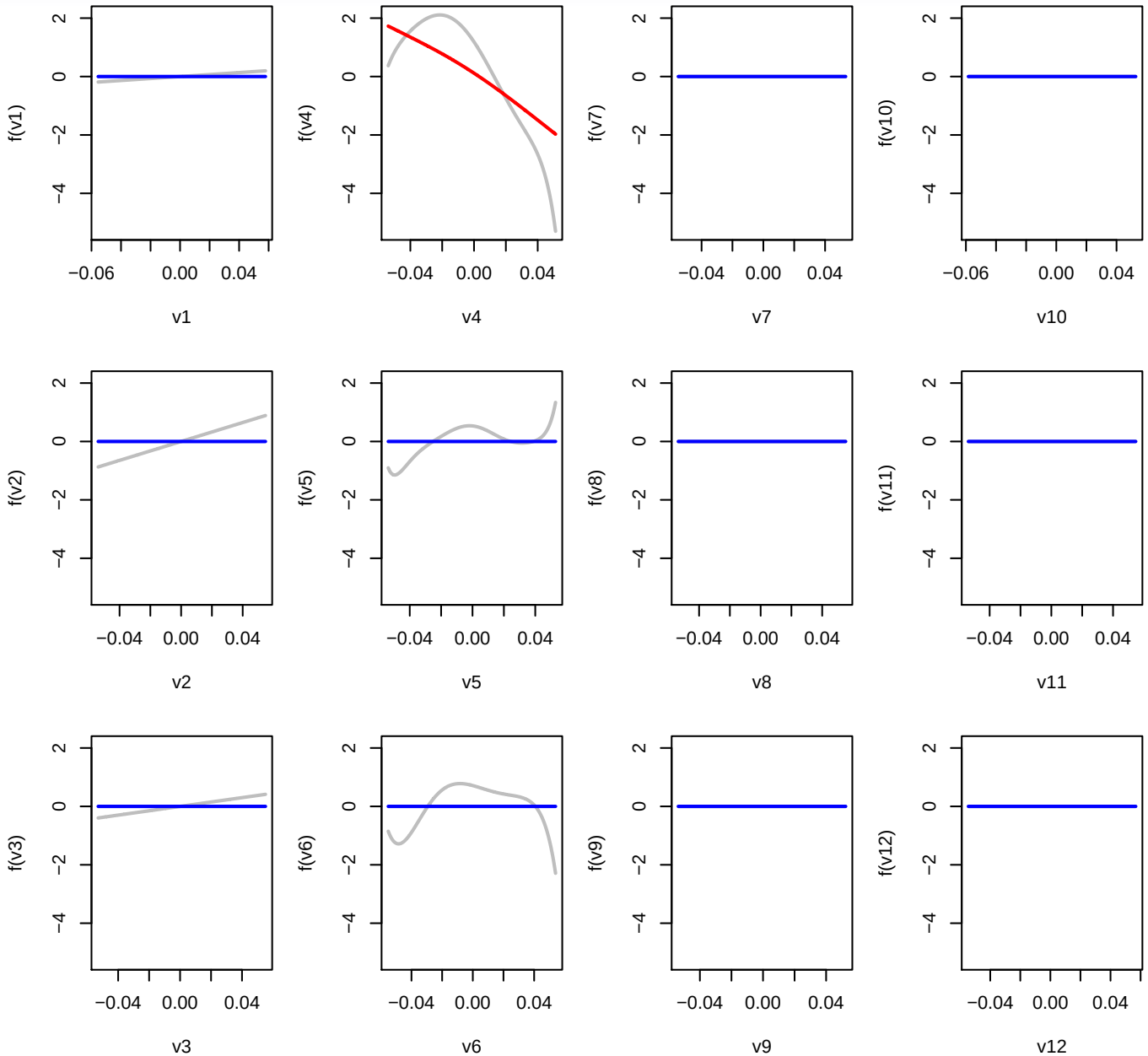

Step= 11 lambda = 49.01
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

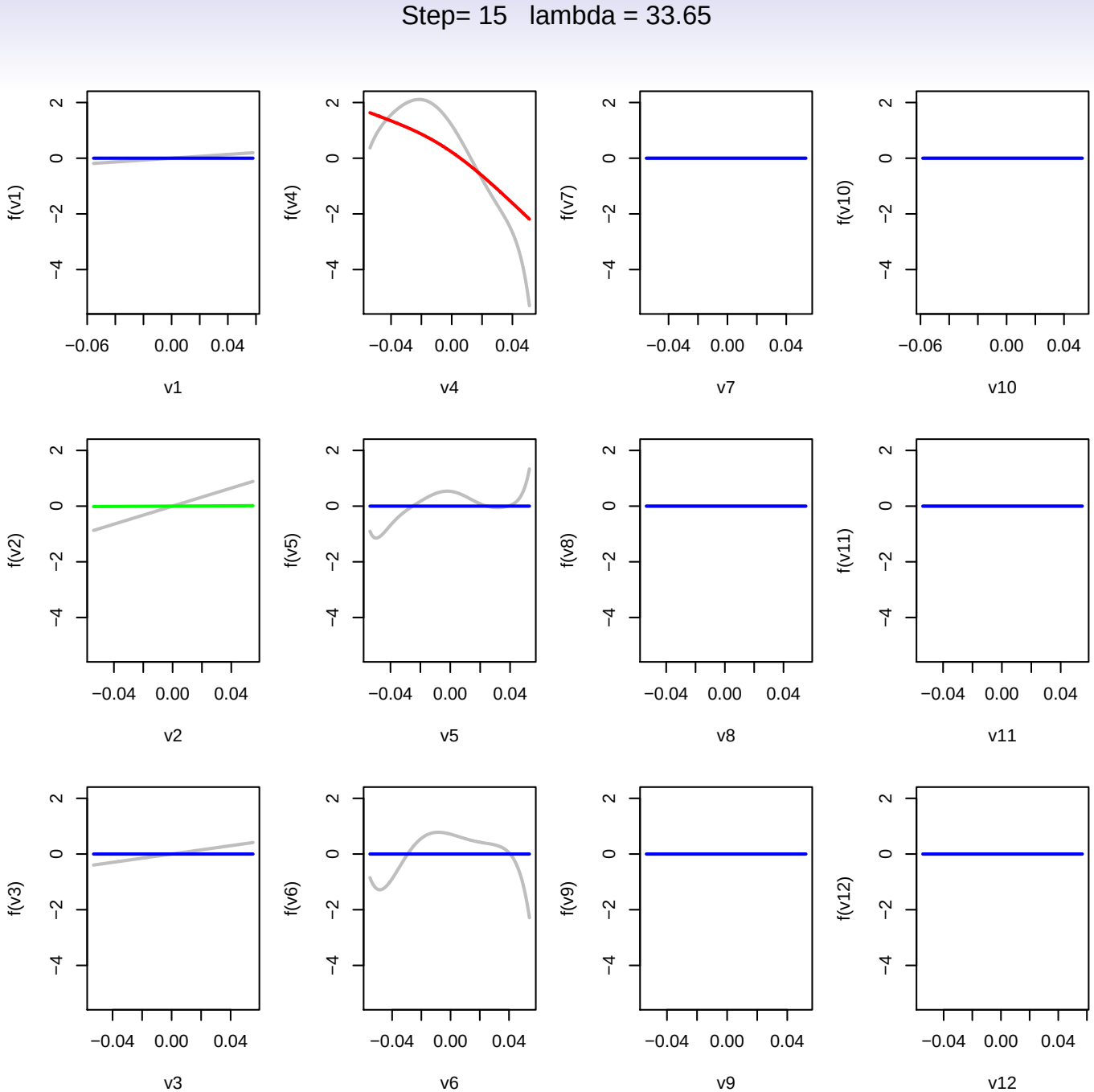

Step= 12 lambda = 44.61
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

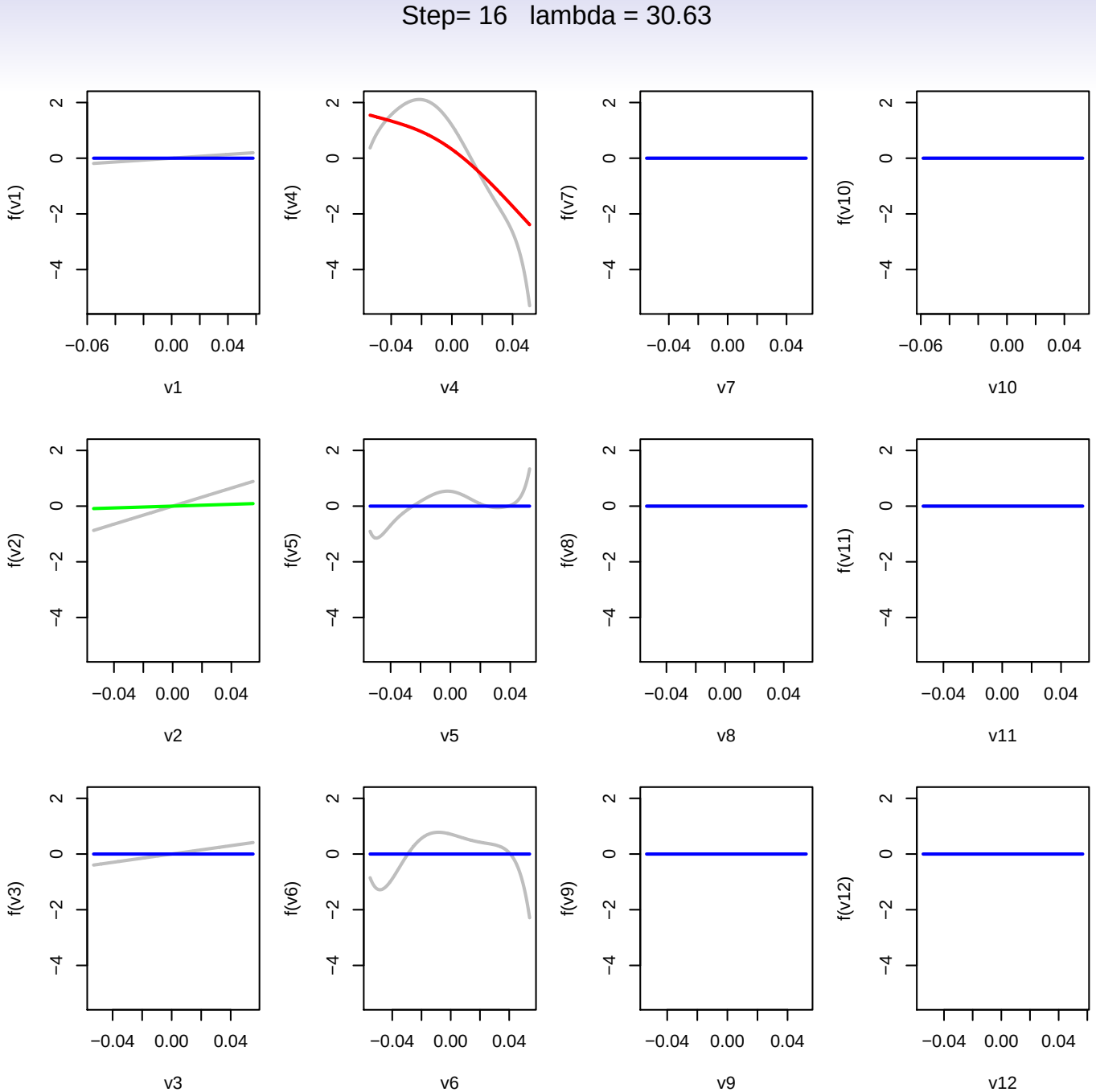
Step= 13 lambda = 40.61

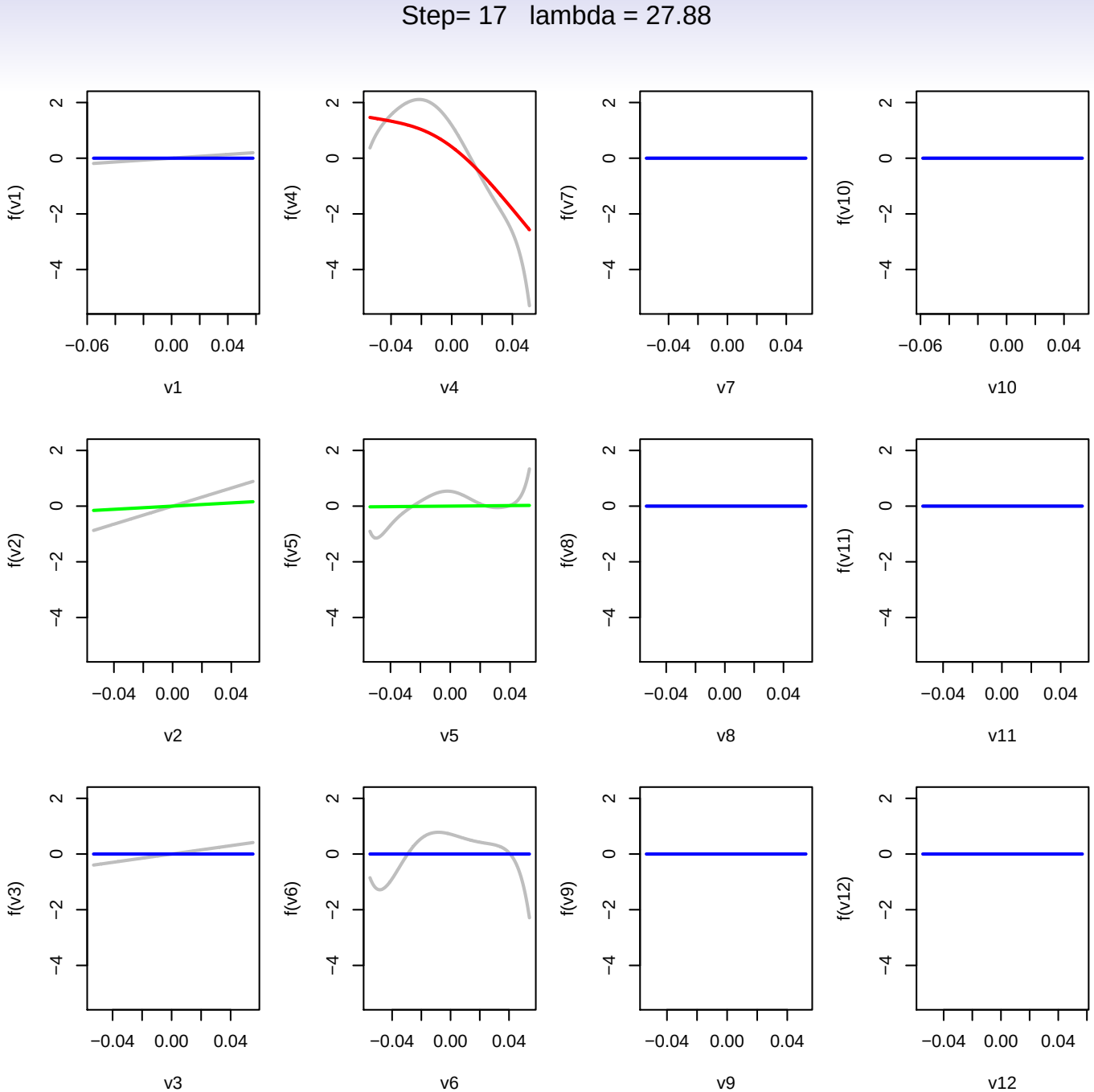


Step= 14 lambda = 36.97

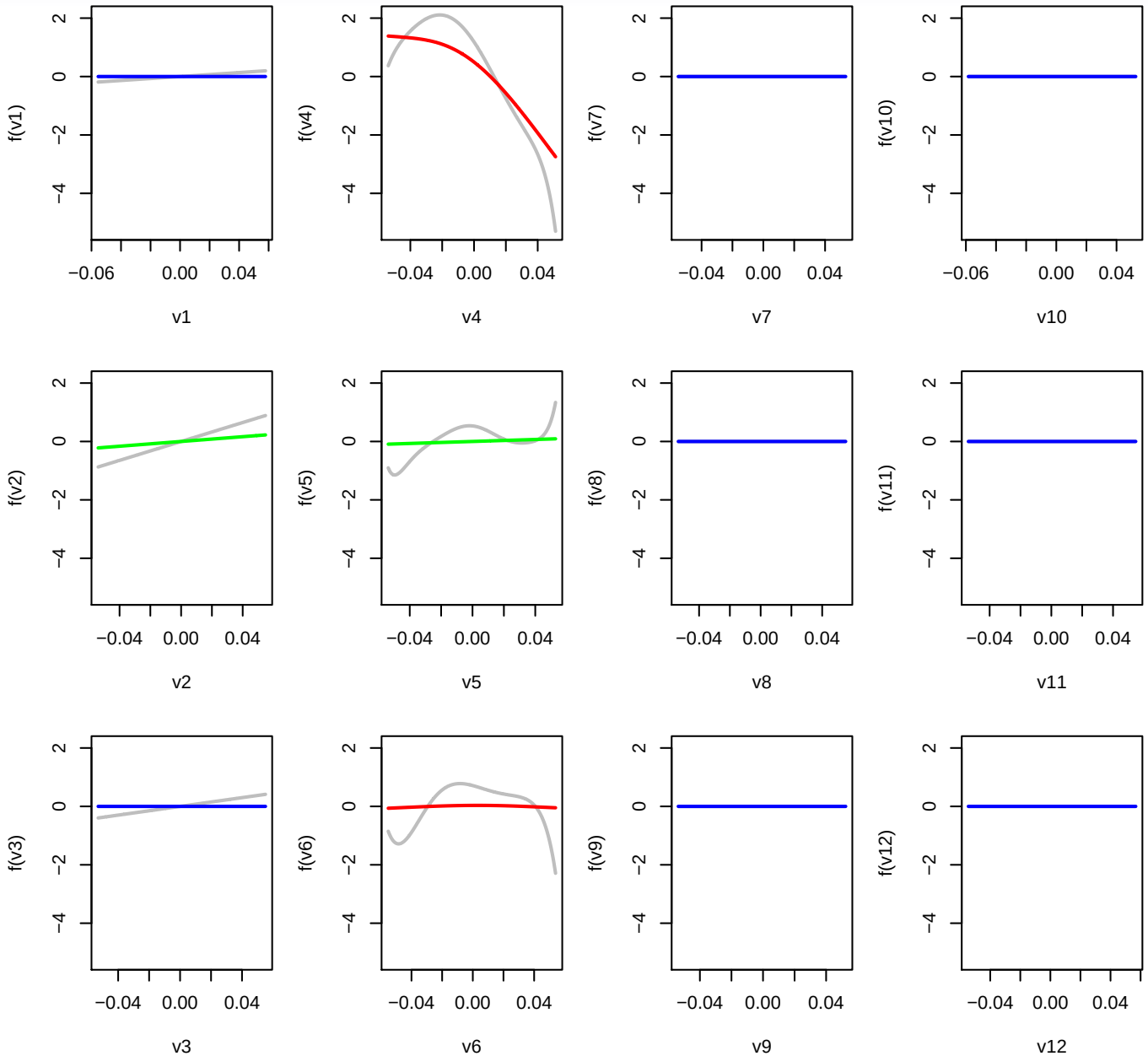


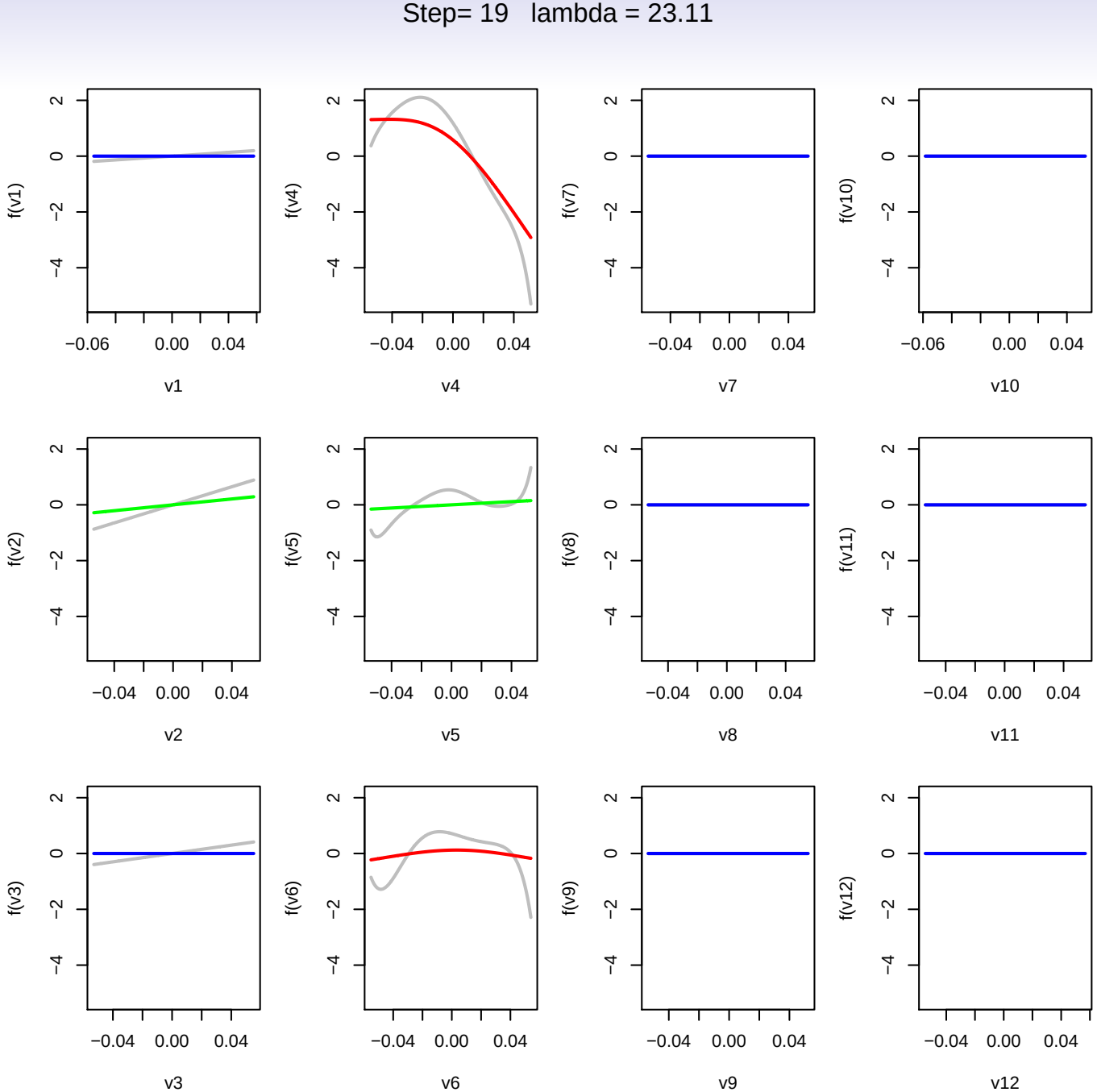

Step= 15 lambda = 33.65
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

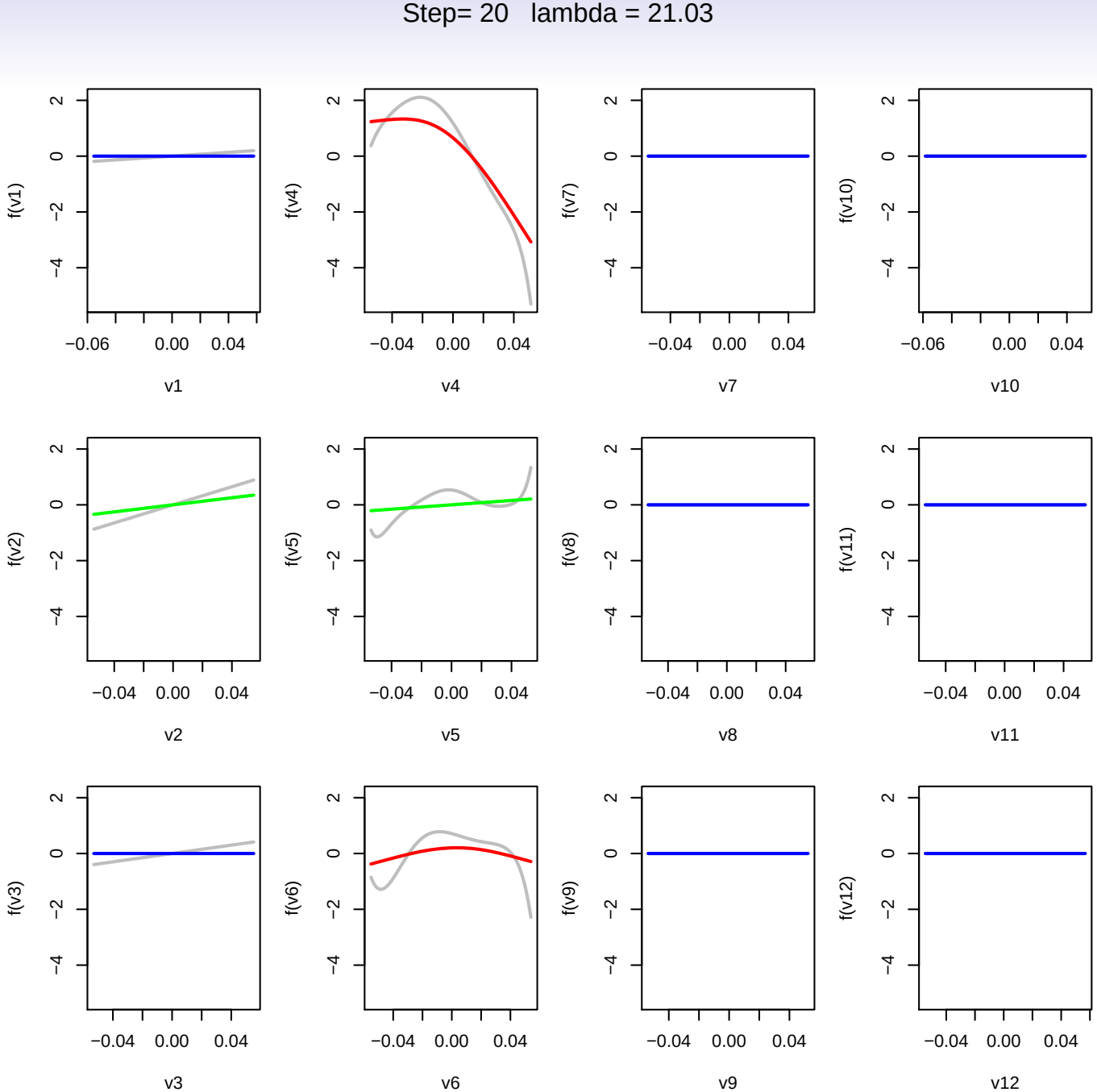

Step= 16 lambda = 30.63
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

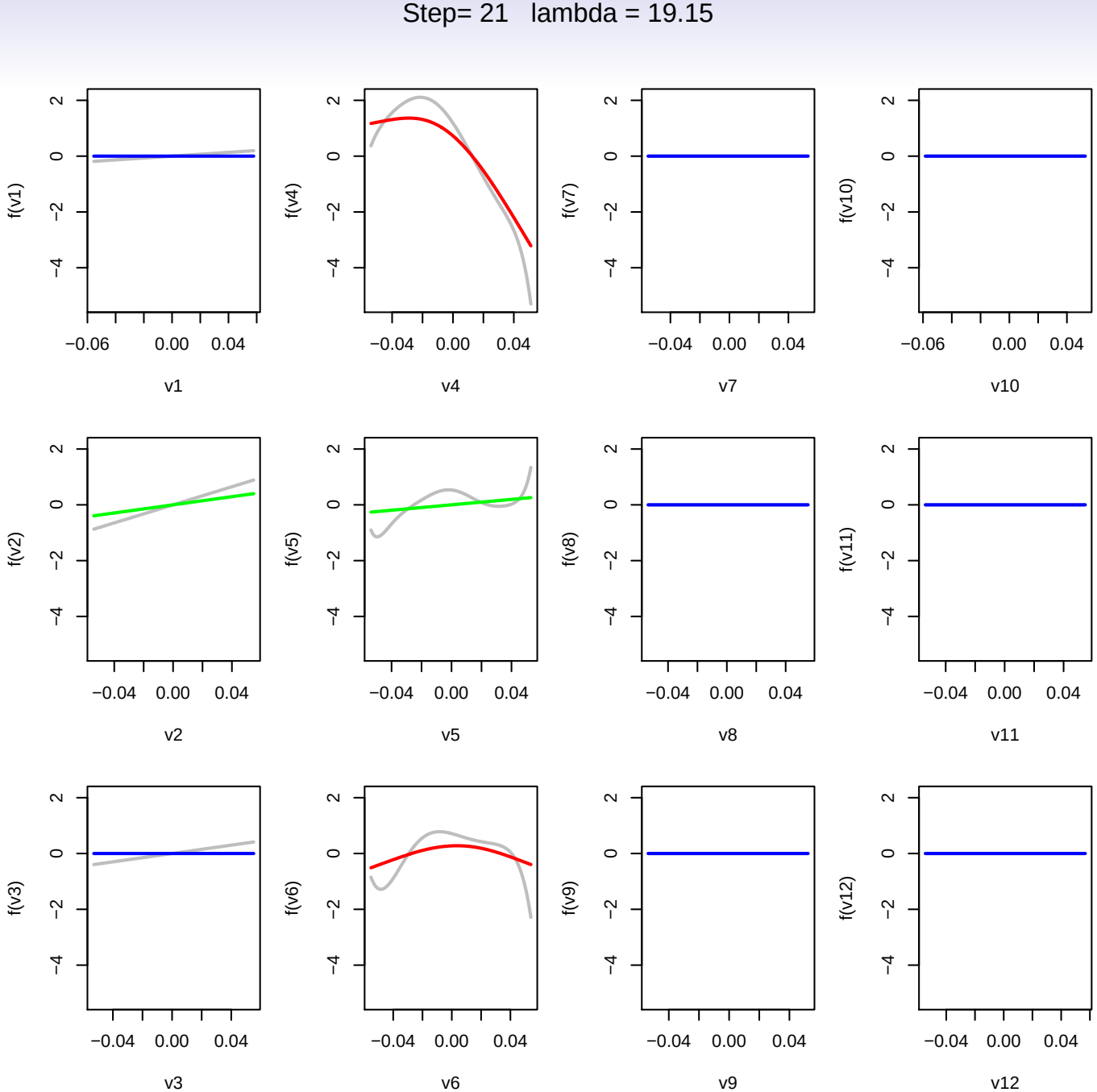

Step= 17 lambda = 27.88
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12

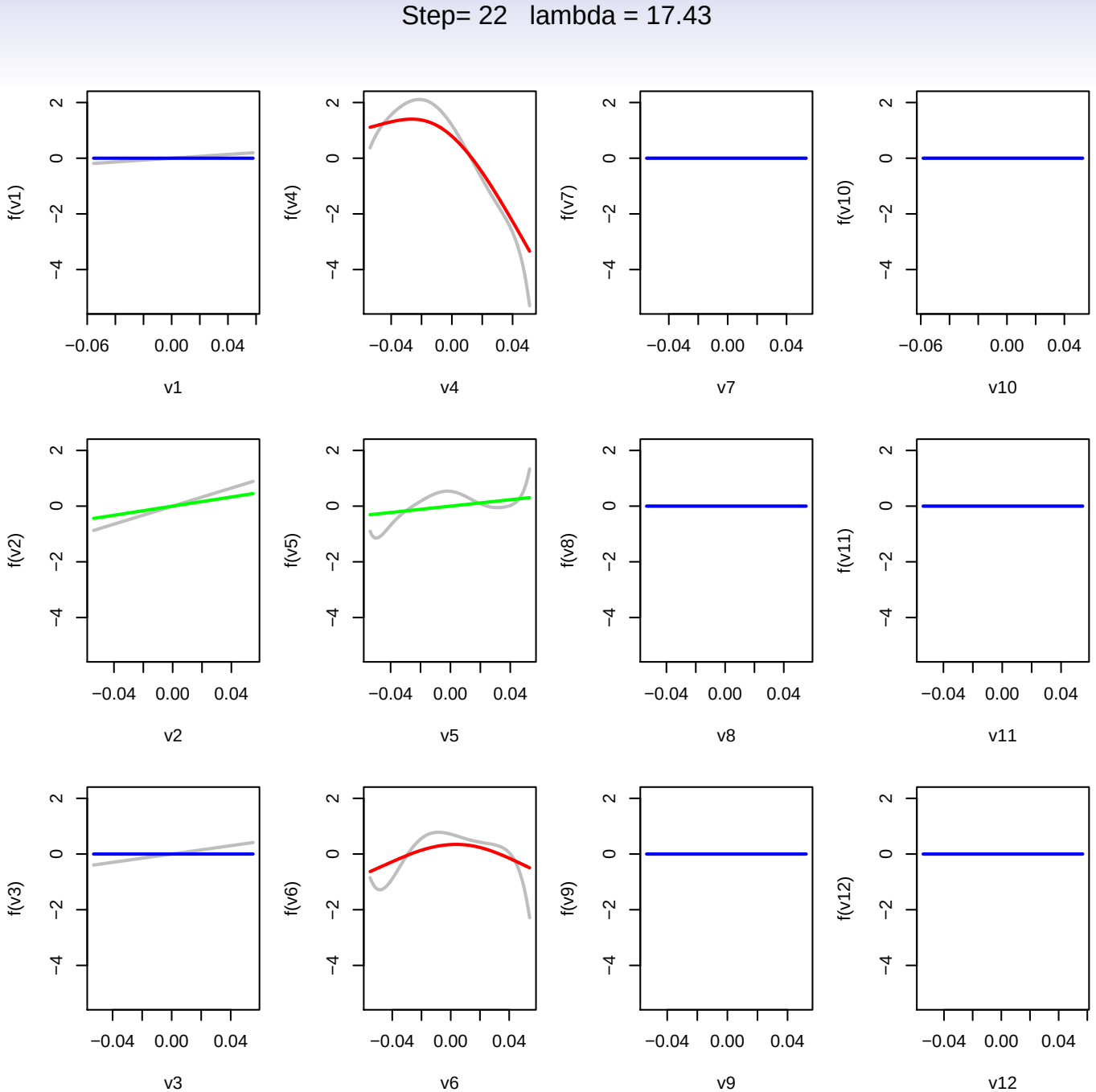
Step= 18 lambda = 25.38



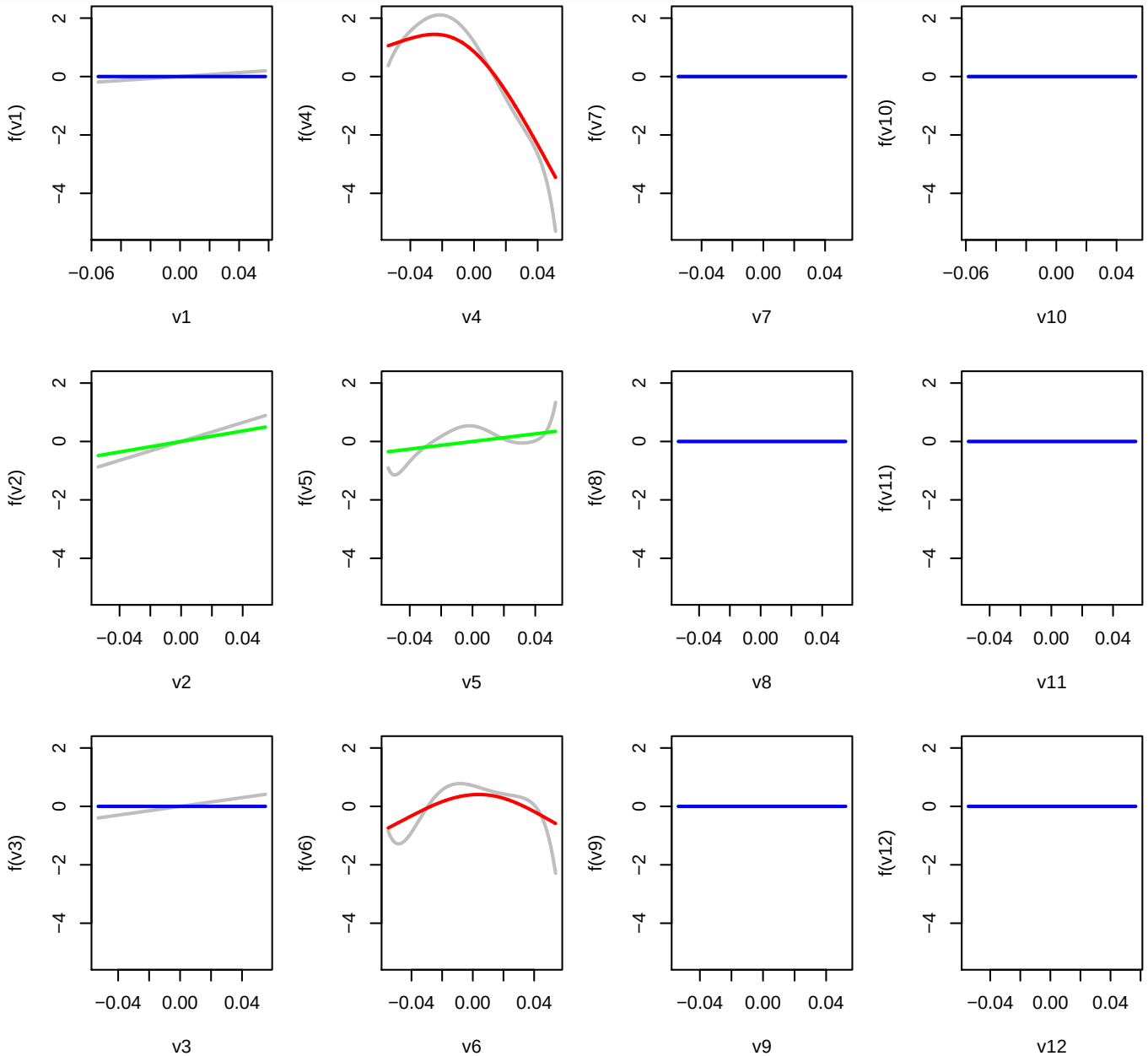

Step= 19 lambda = 23.11
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12

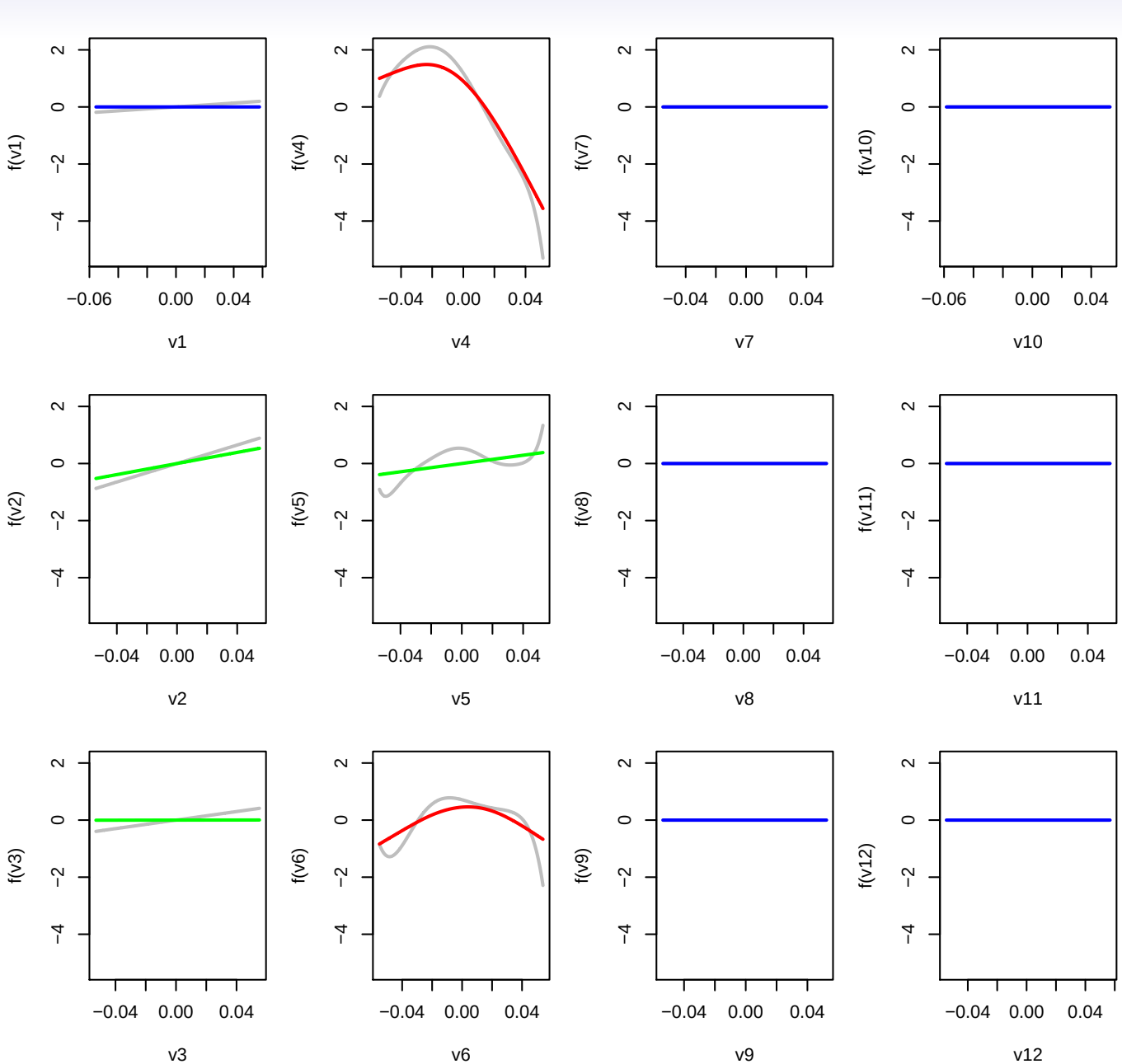

Step= 20 lambda = 21.03
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12

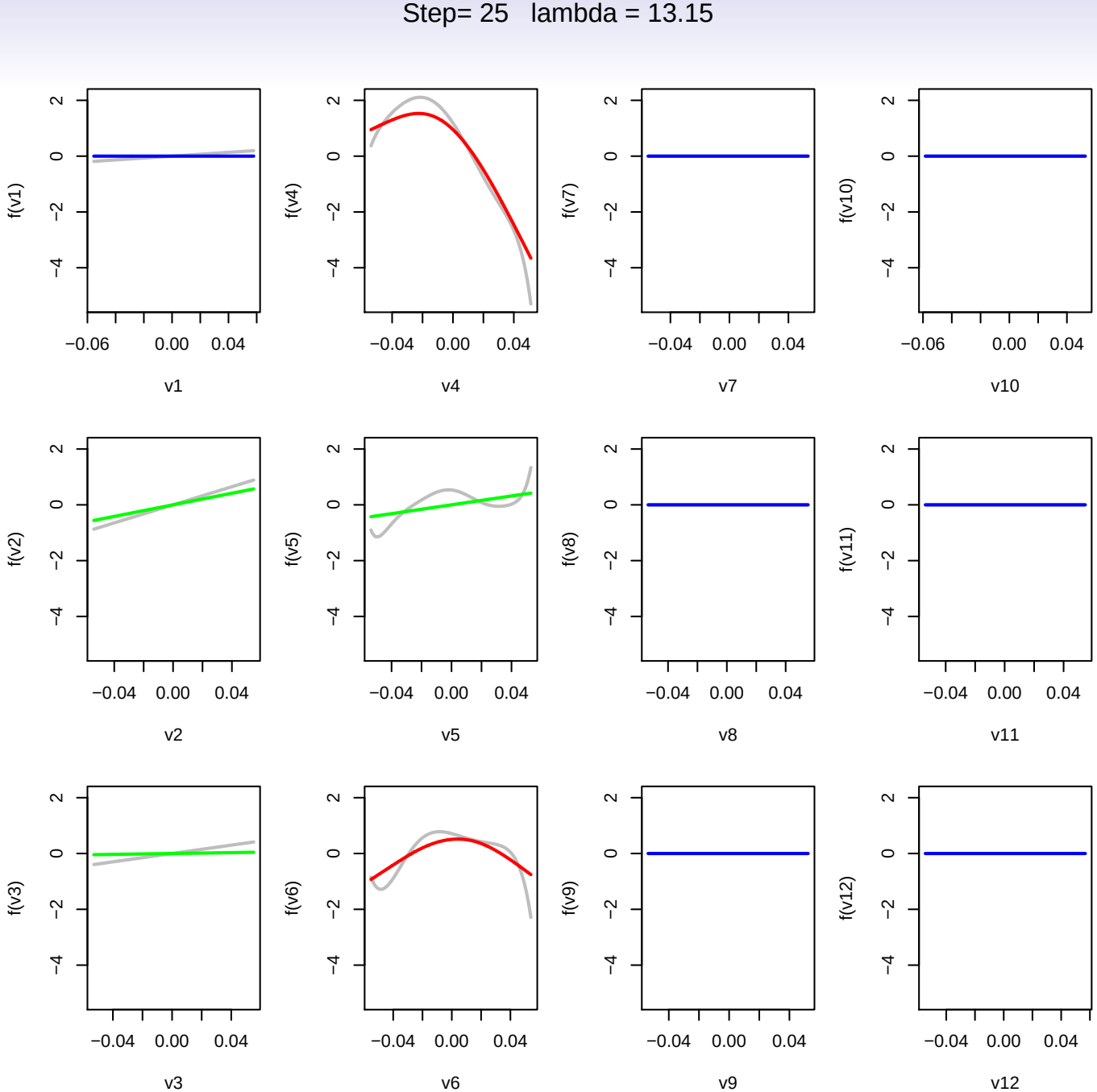

Step= 21 lambda = 19.15
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

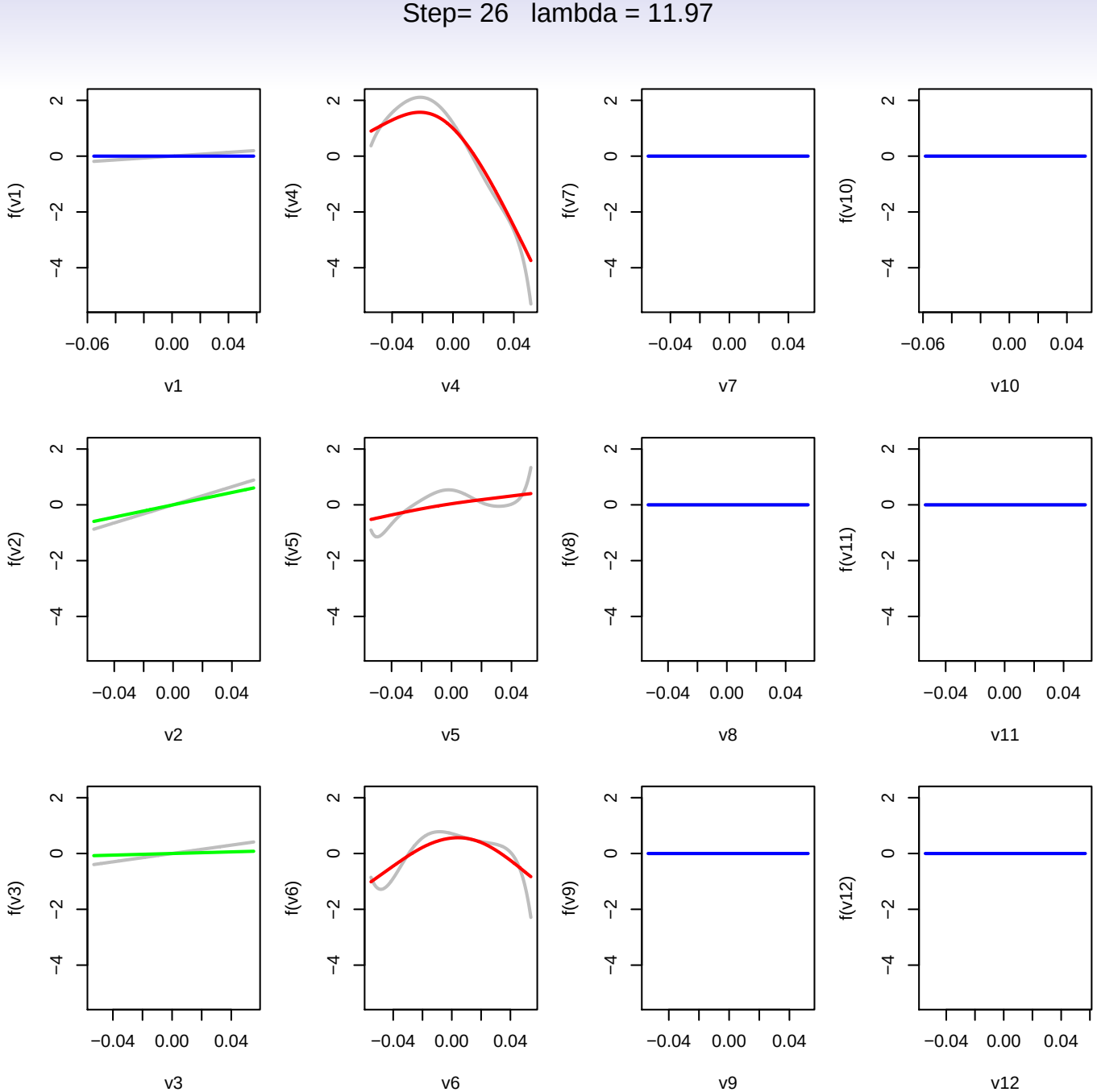

Step= 22 lambda = 17.43
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

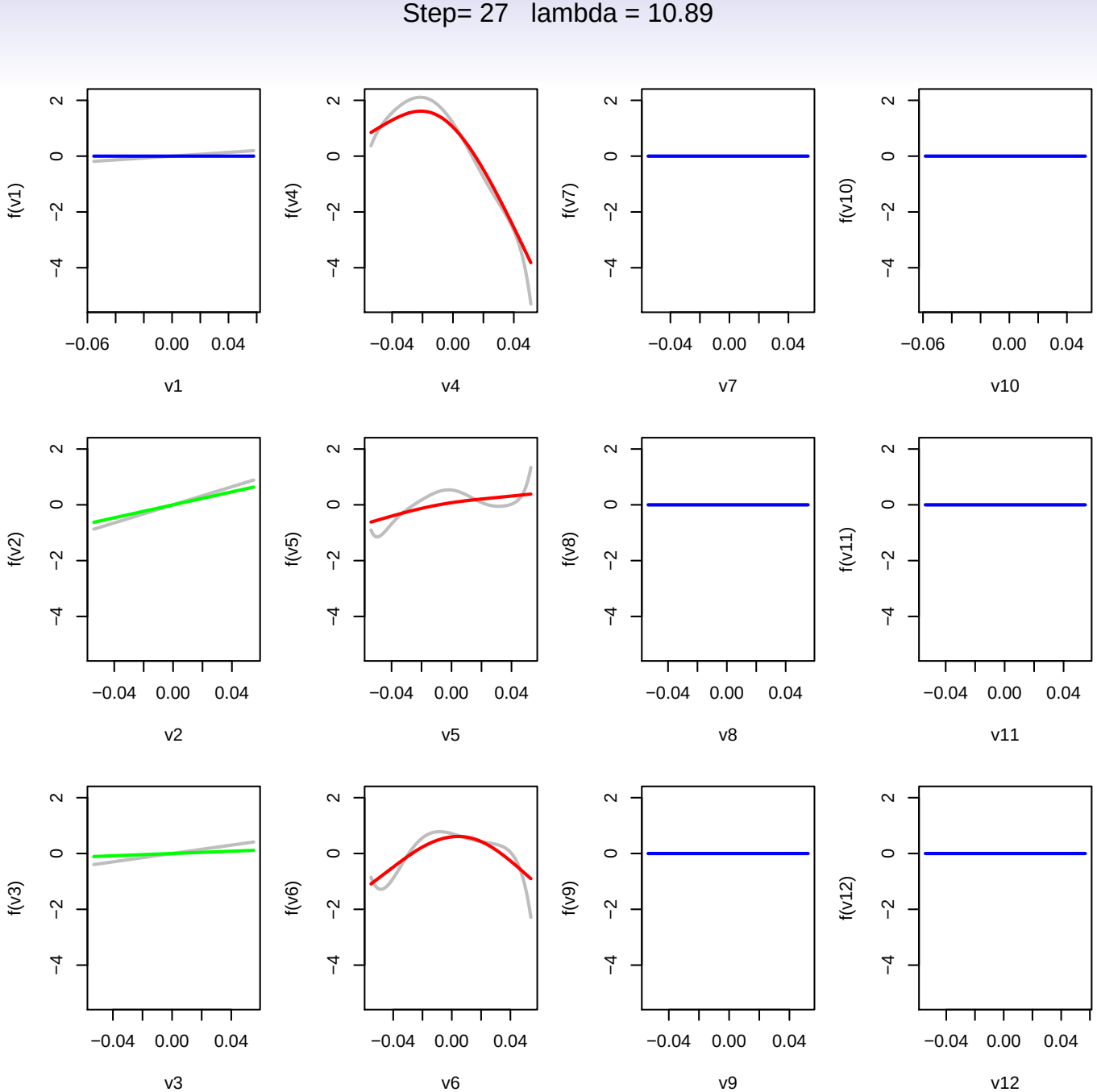
Step= 23 lambda = 15.87

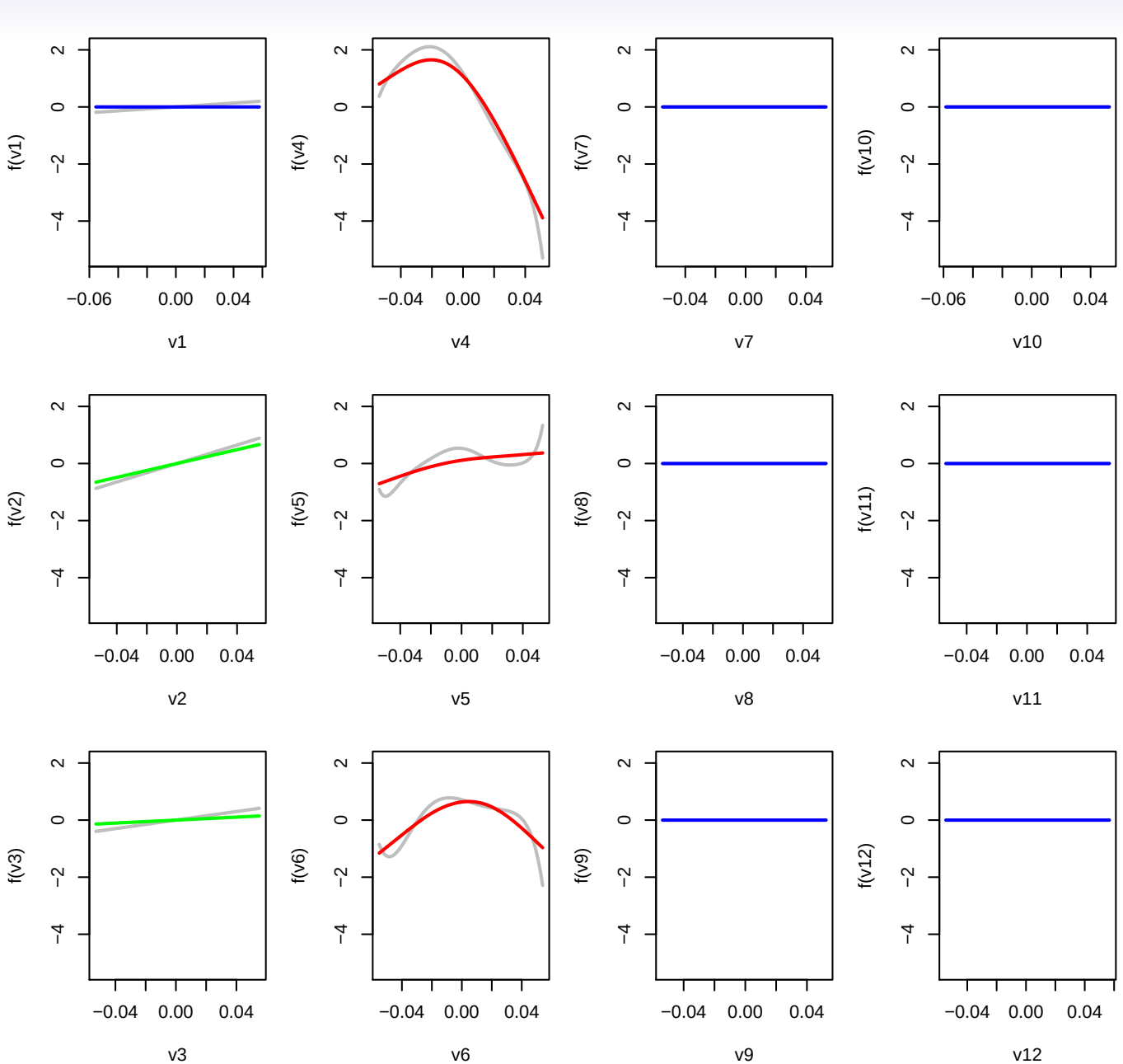


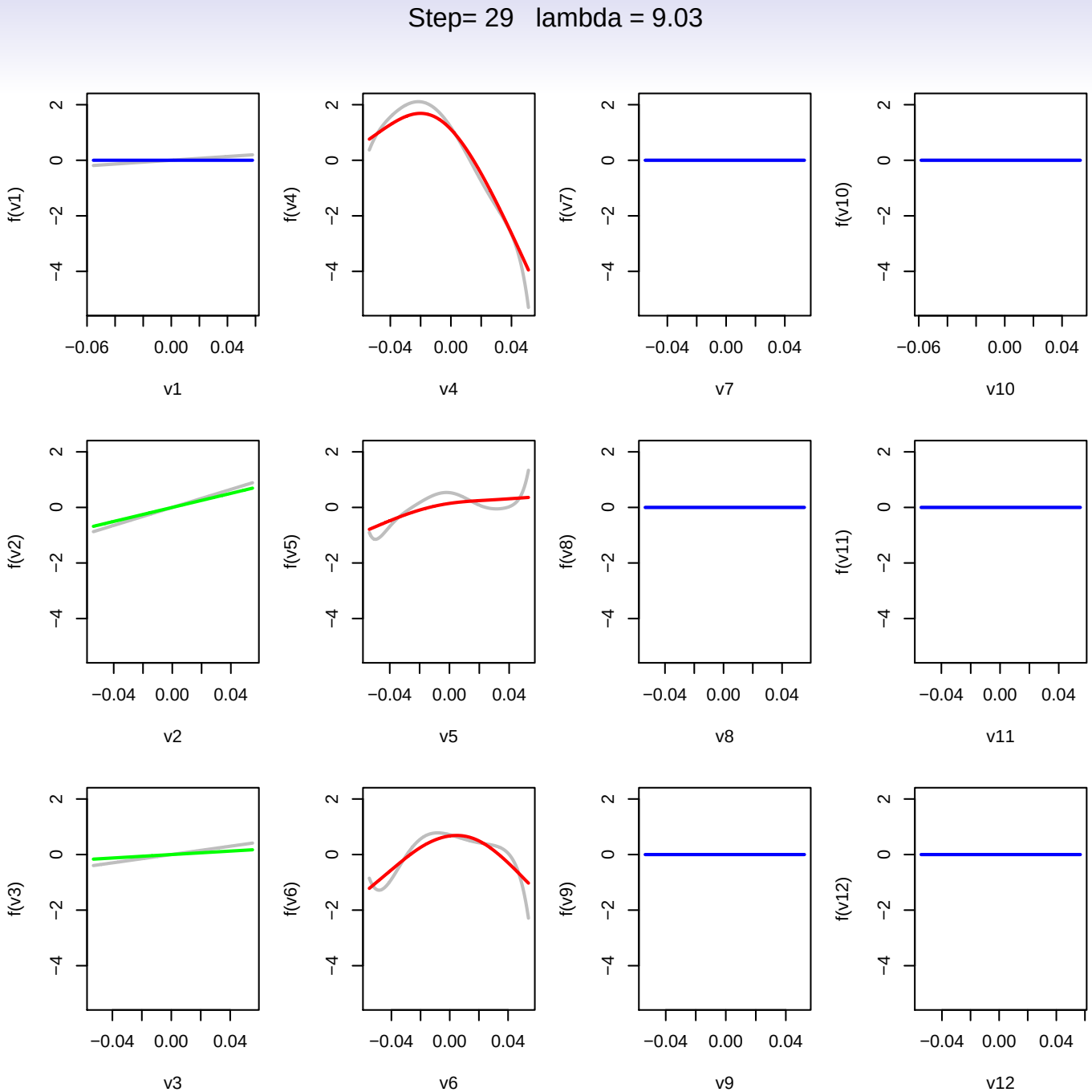

Step= 24 lambda = 14.44
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12


Step= 25 lambda = 13.15
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

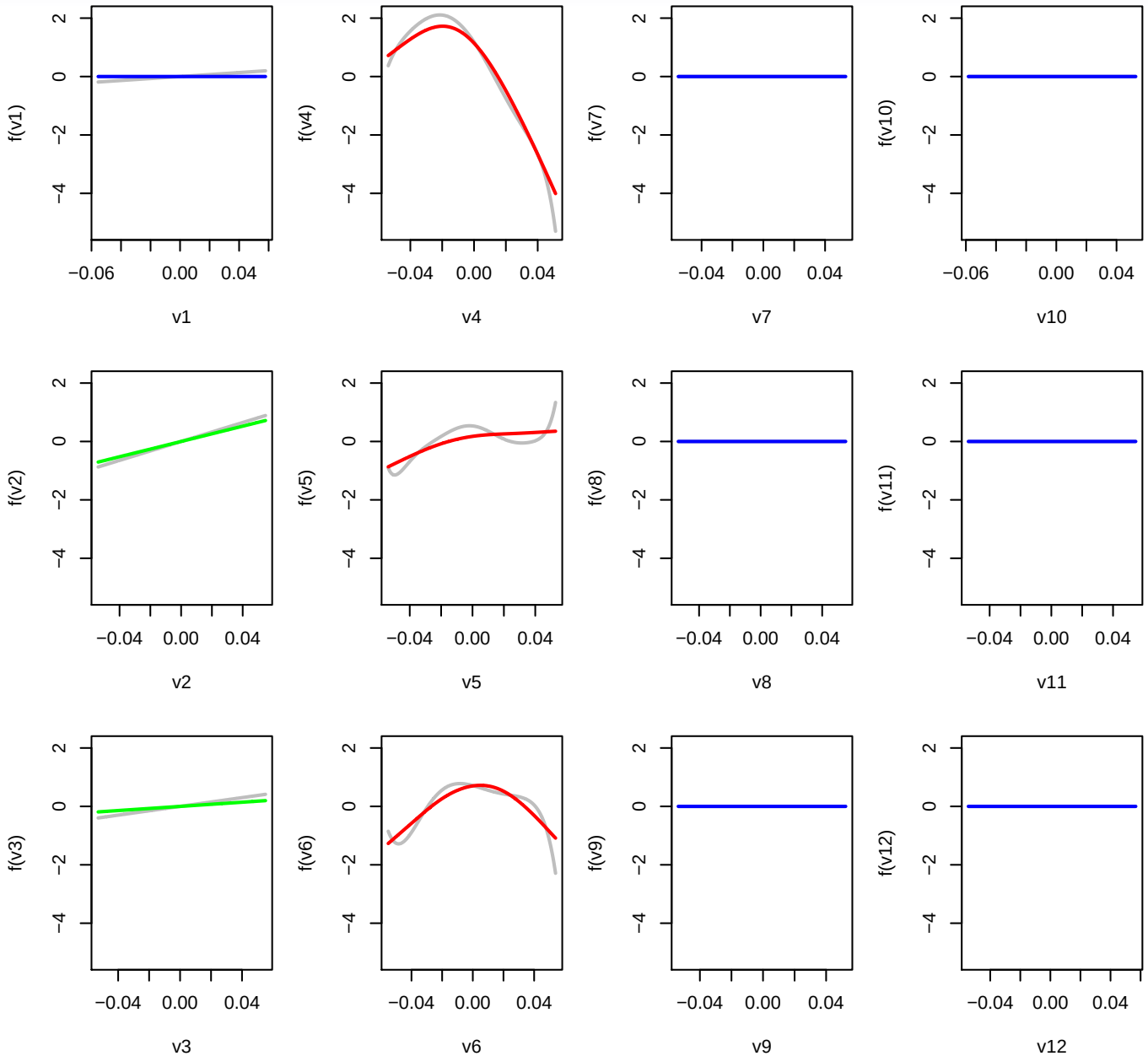

Step= 26 lambda = 11.97
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12

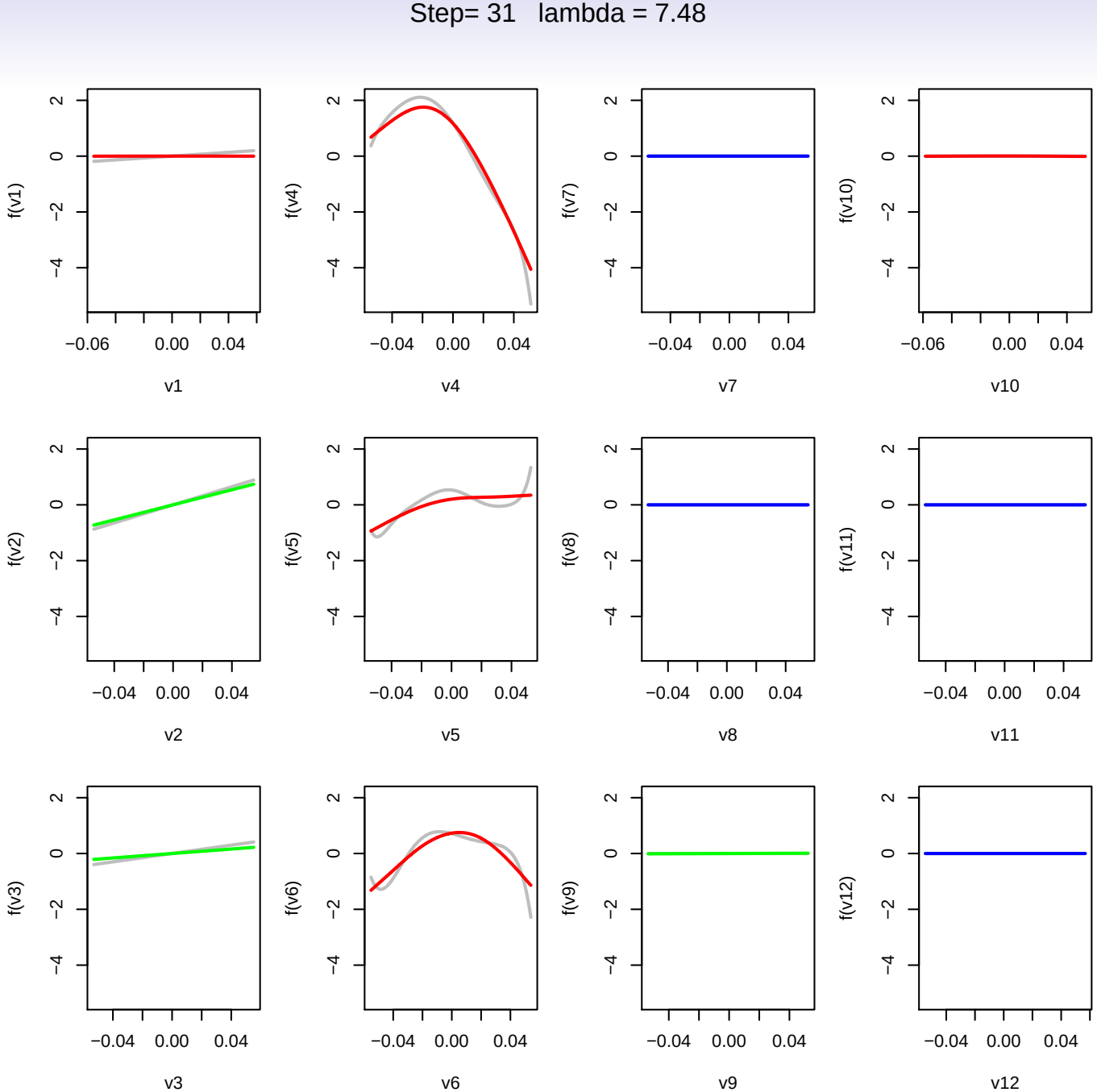

Step= 27 lambda = 10.89
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

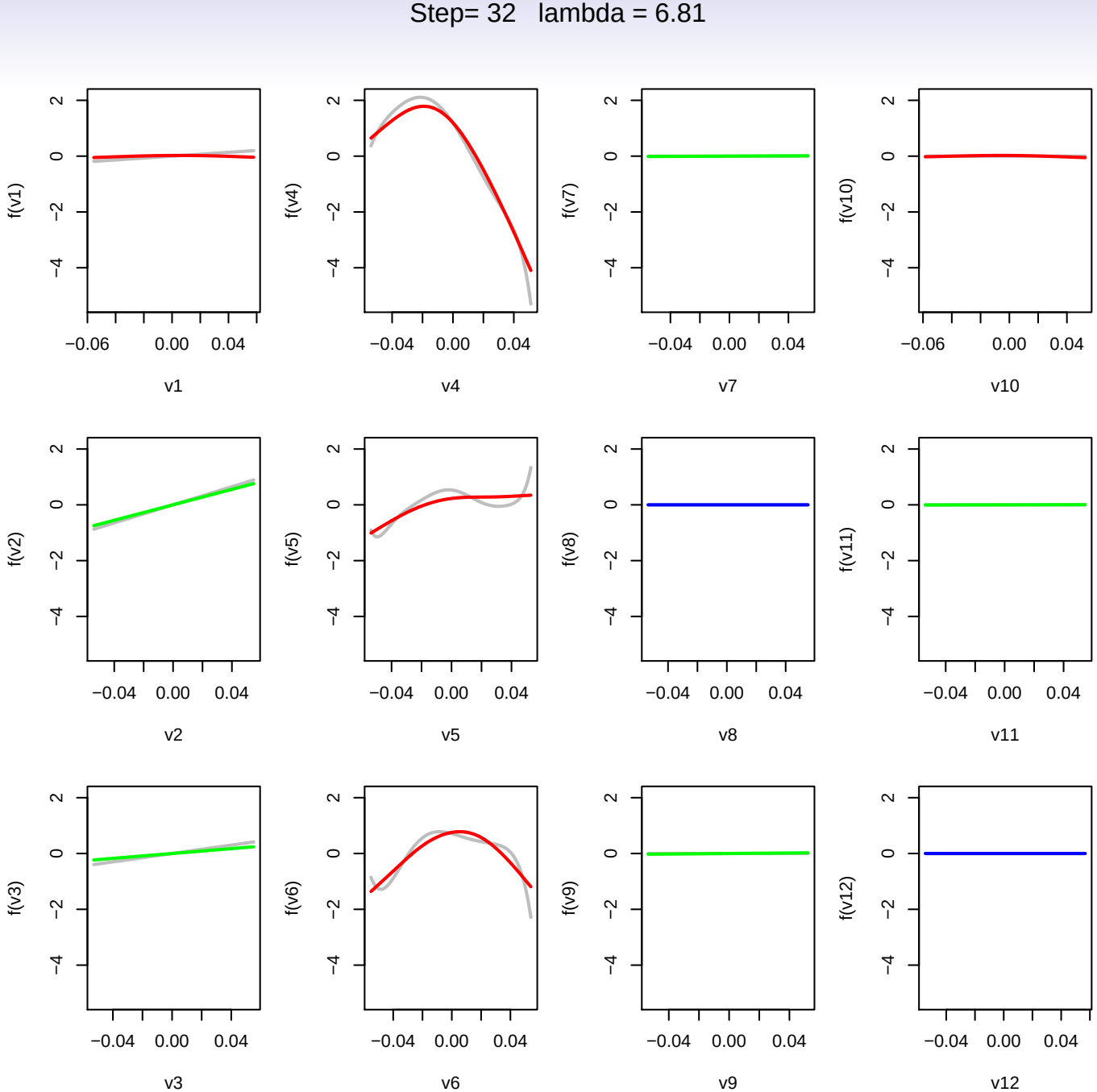

Step= 28 lambda = 9.92
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12


Step= 29 lambda = 9.03
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12
2
0
−2
−4
−0.06
−0.04
0.00
0.04

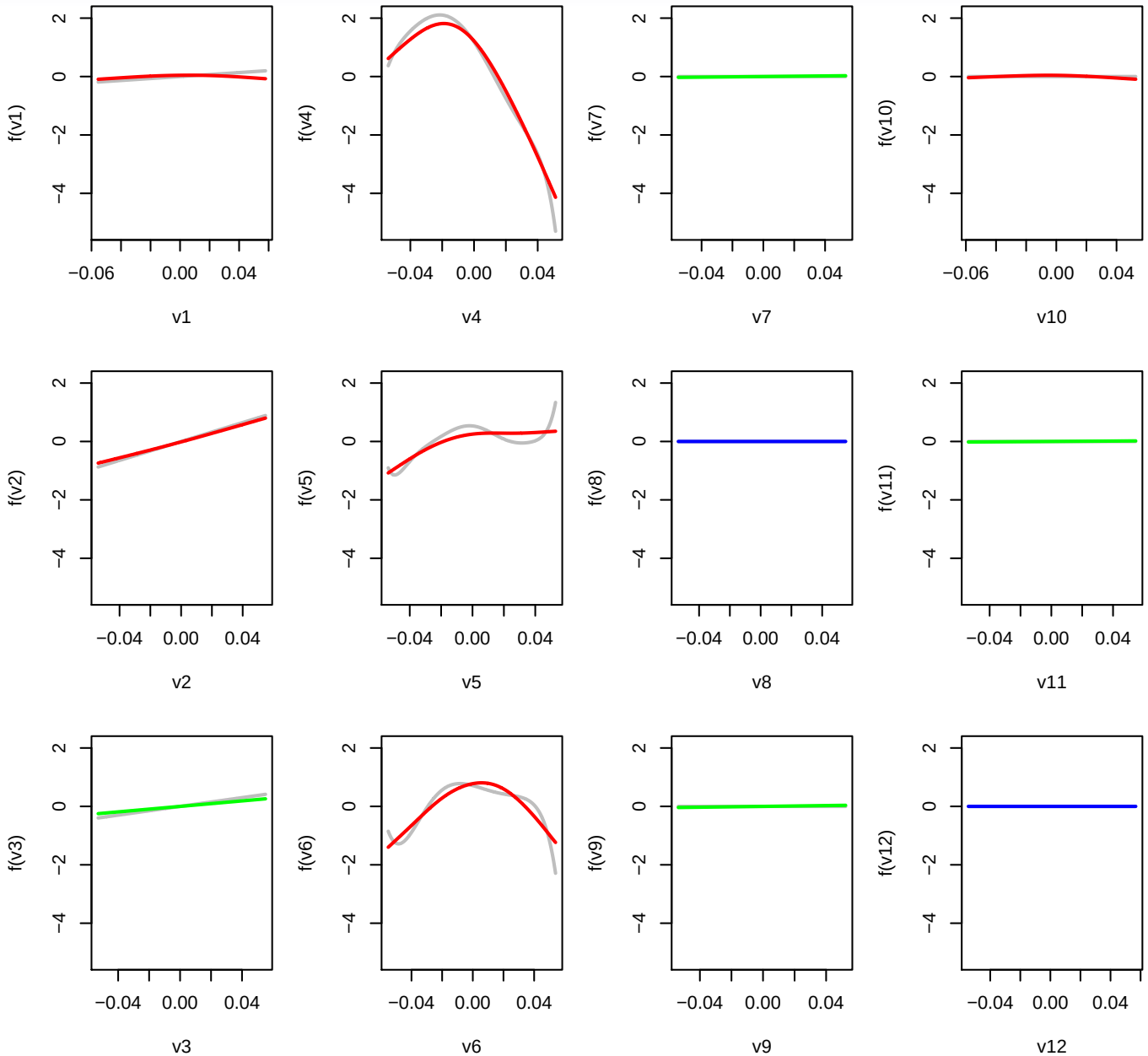
Step= 30 lambda = 8.22

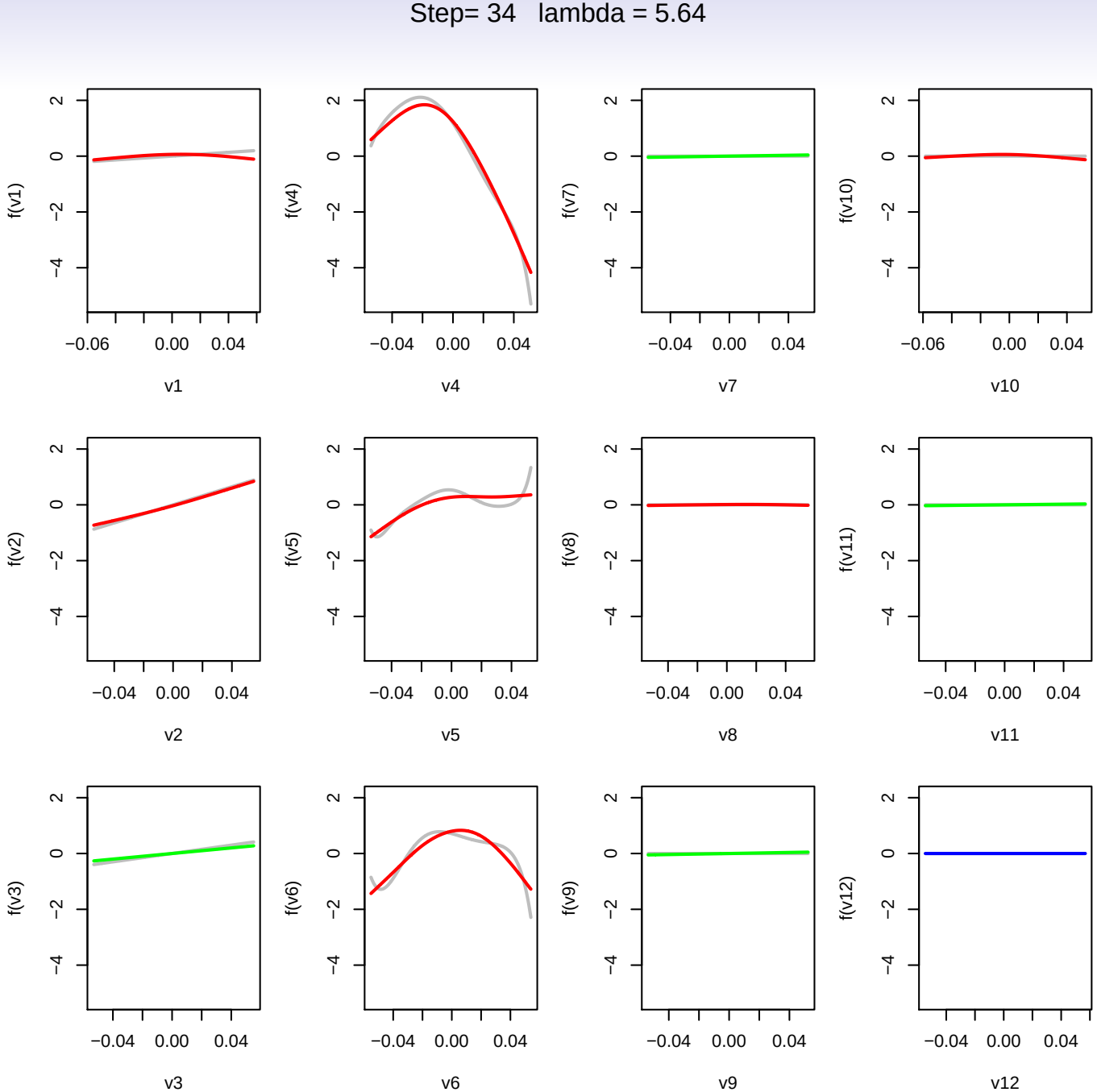


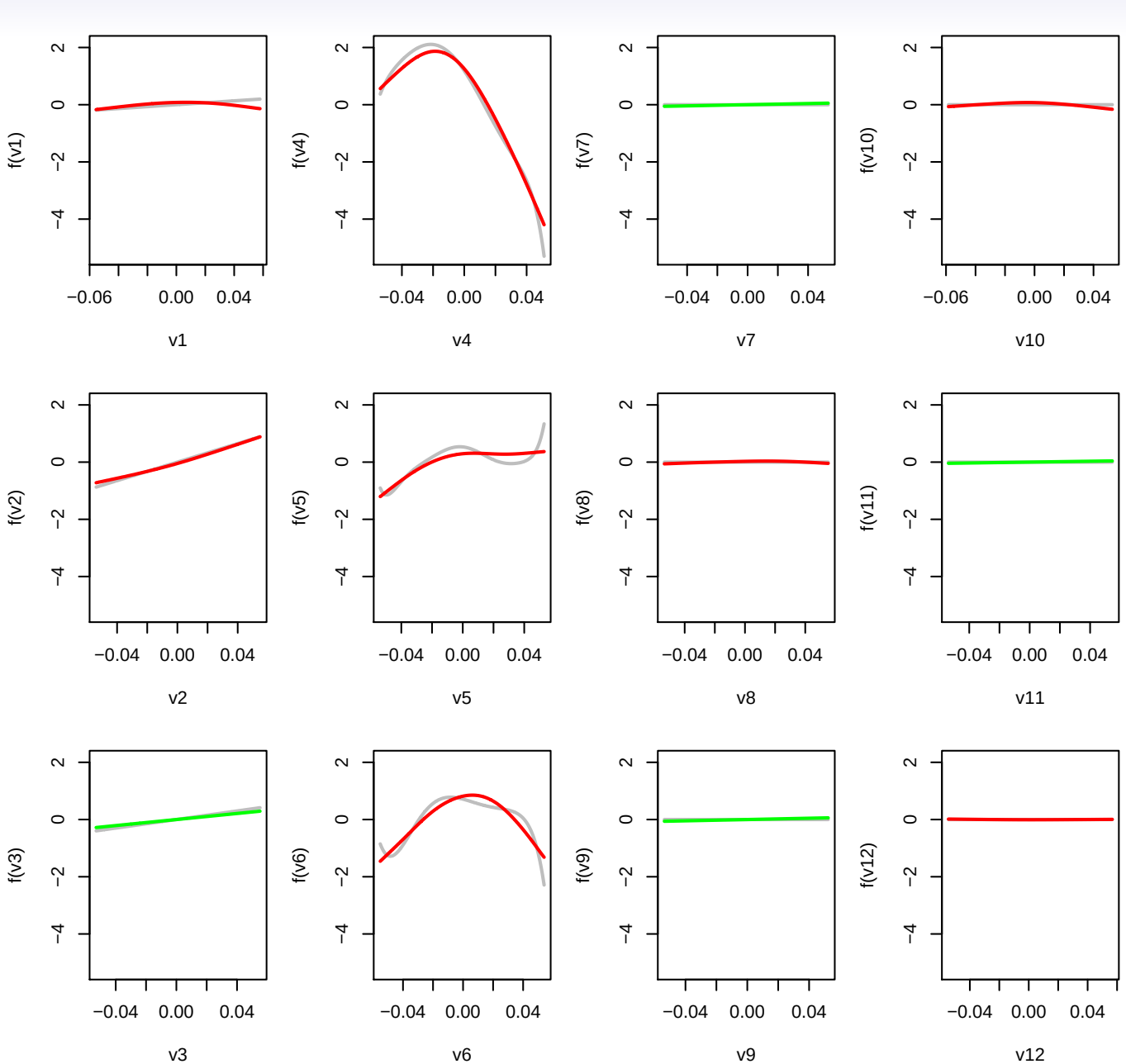

Step= 31 lambda = 7.48
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

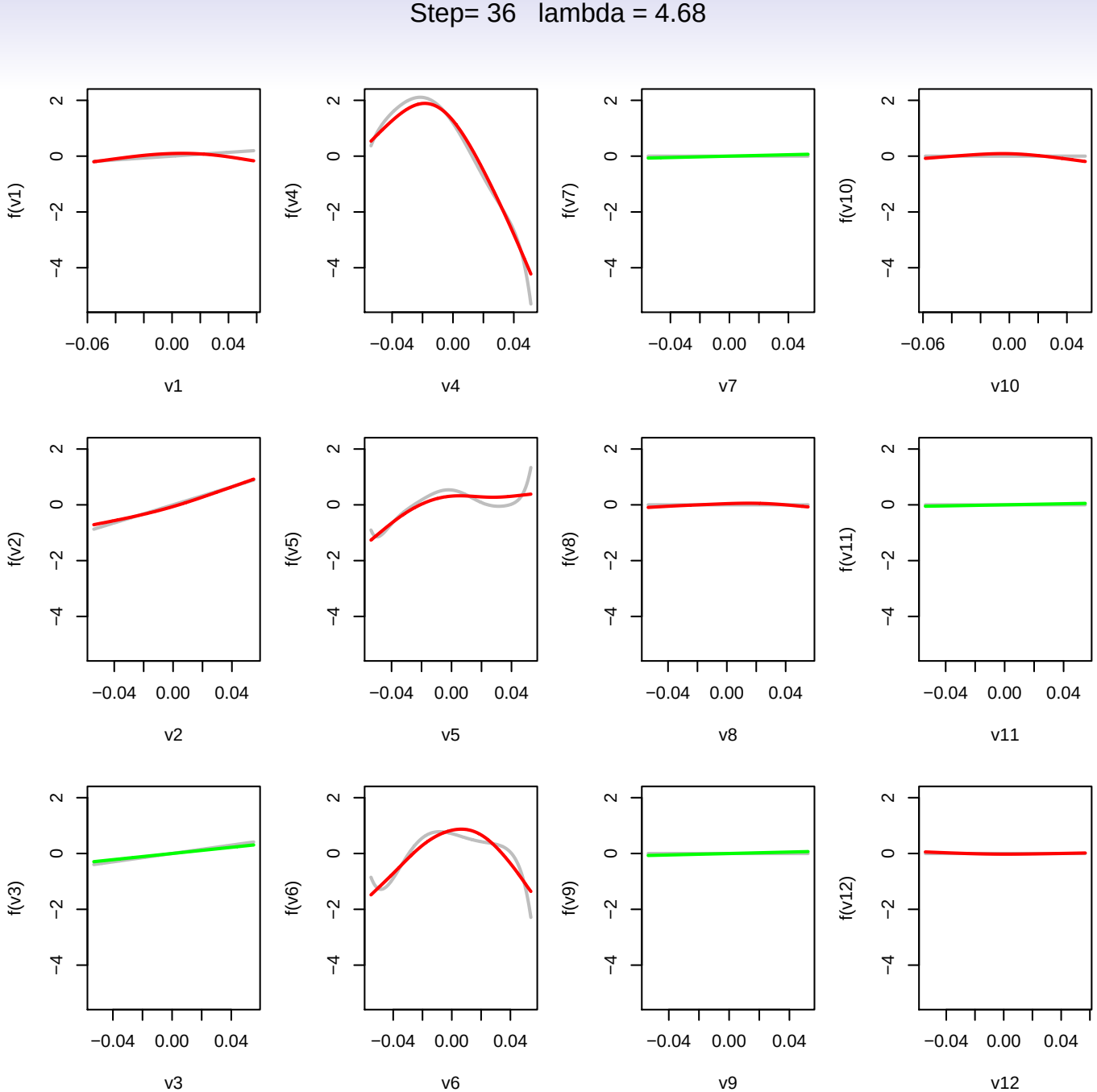

Step= 32 lambda = 6.81
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

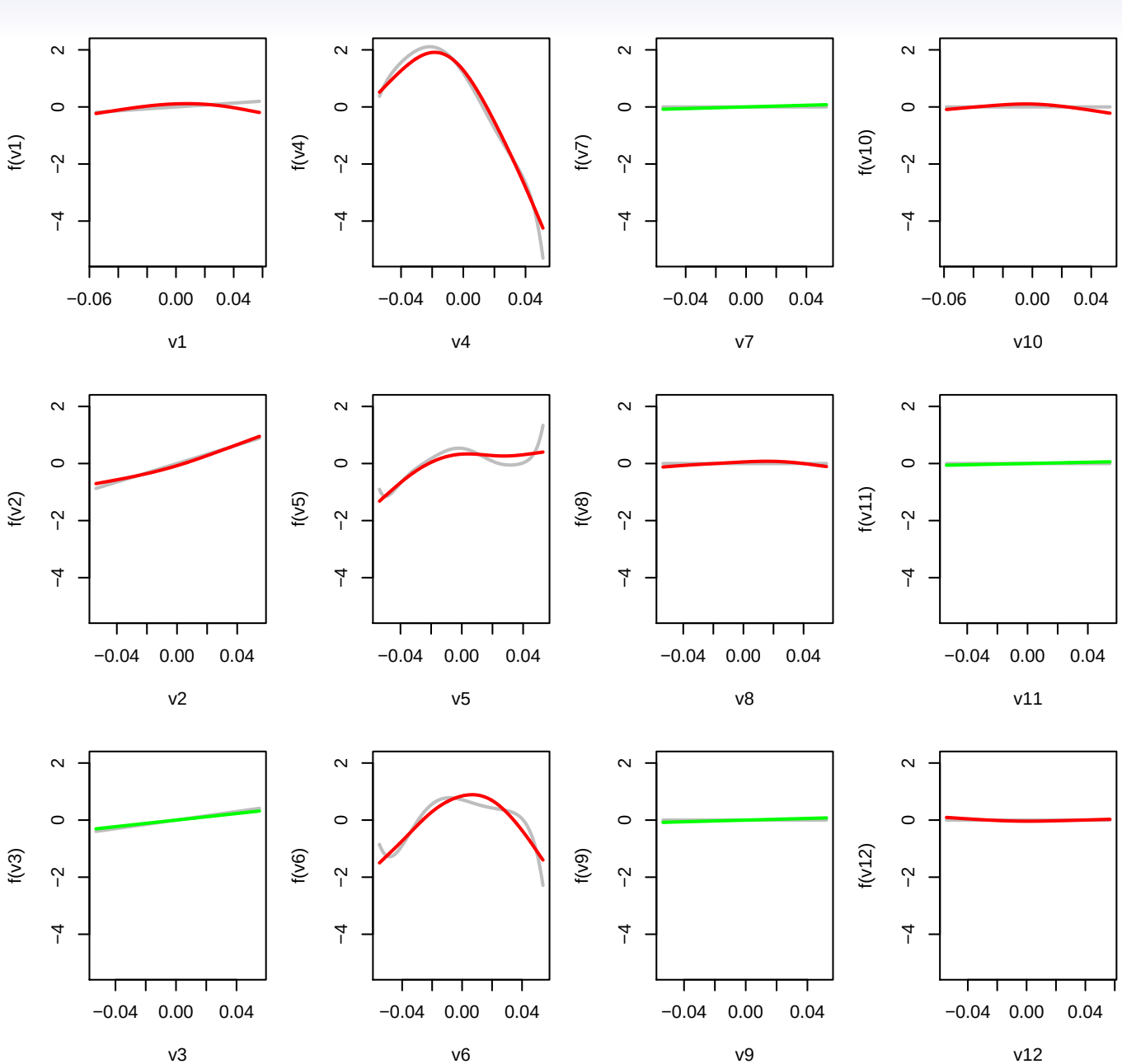
Step= 33 lambda = 6.2

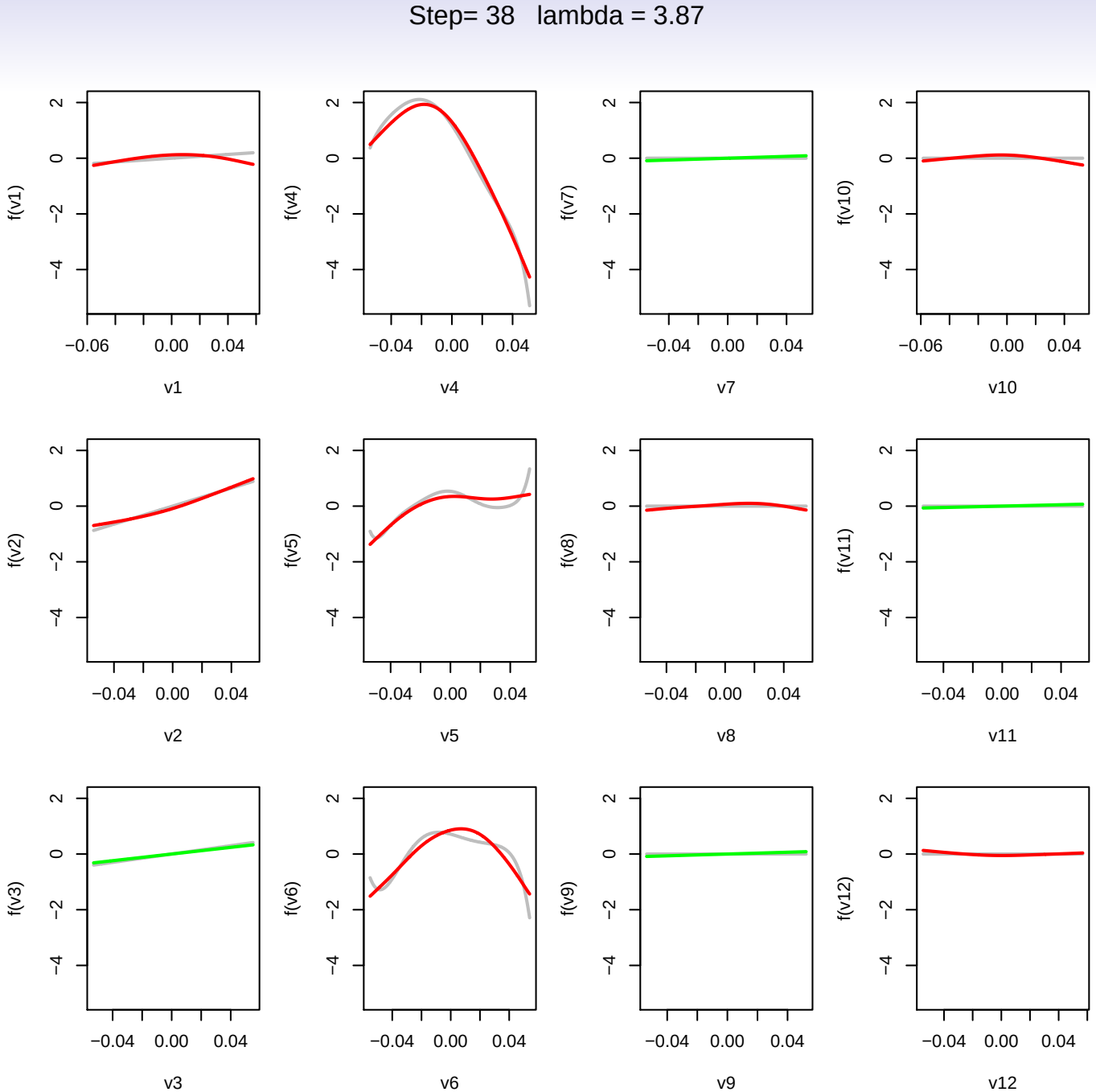


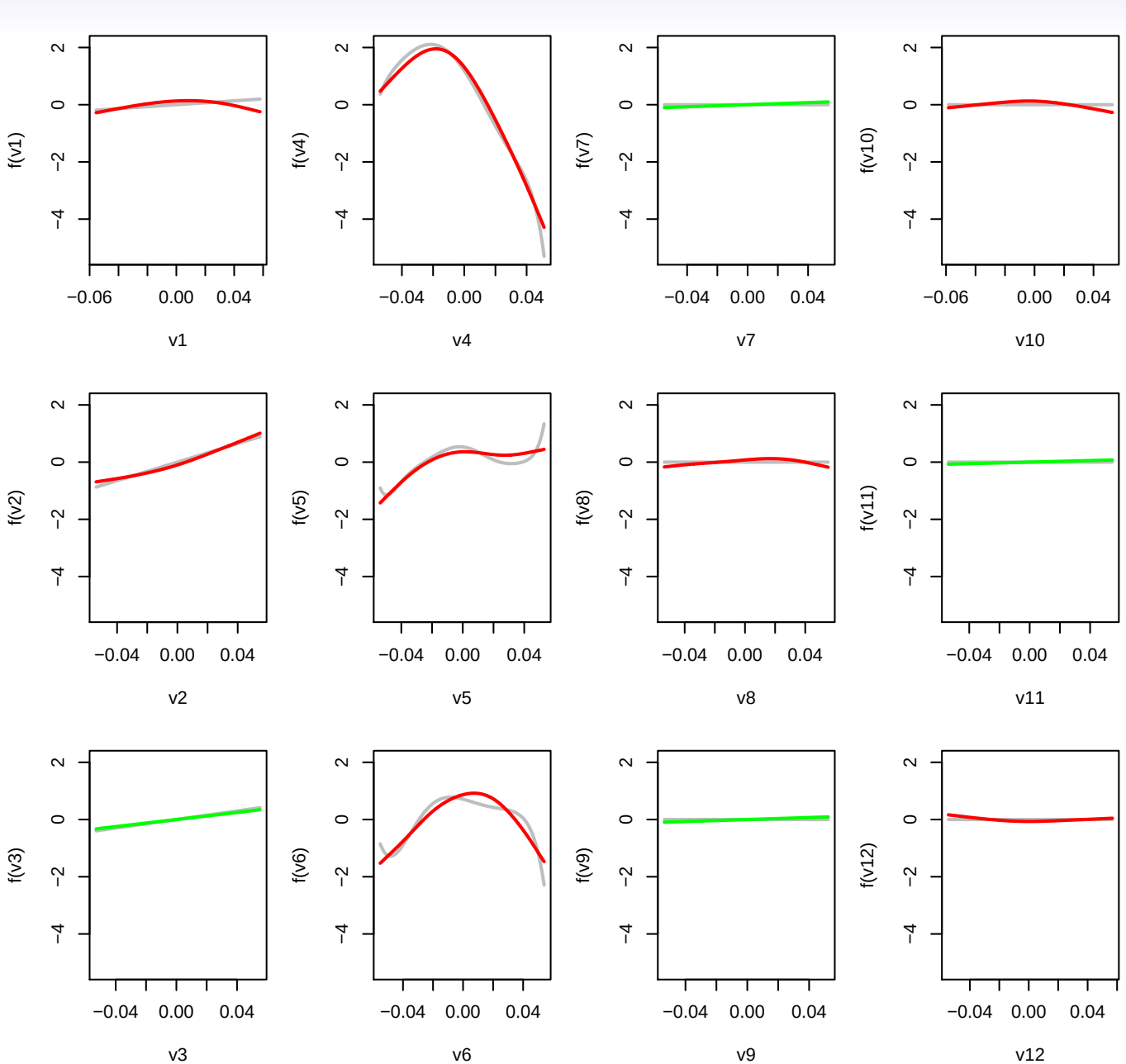

Step= 34 lambda = 5.64
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

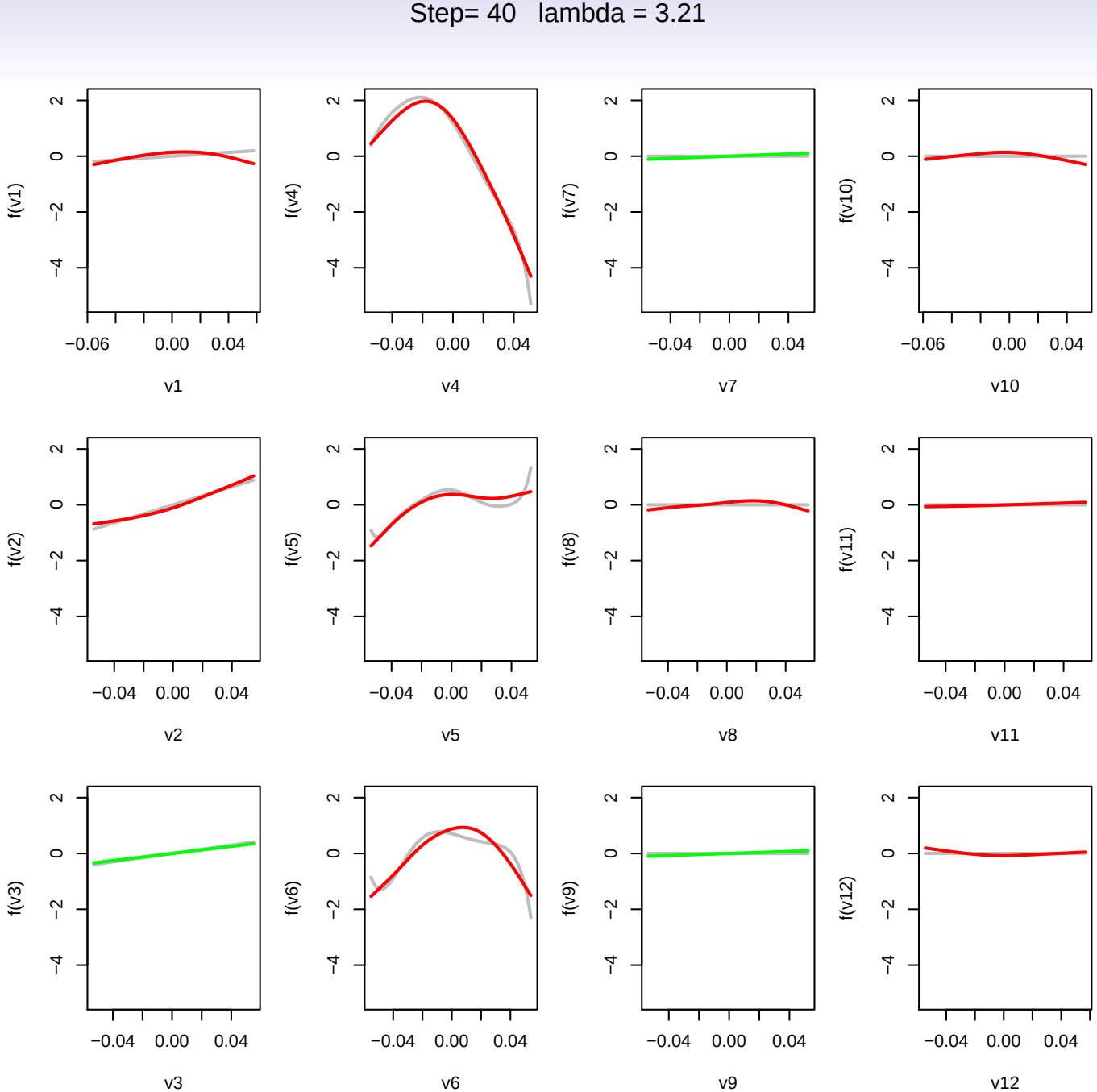

Step= 35 lambda = 5.14
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

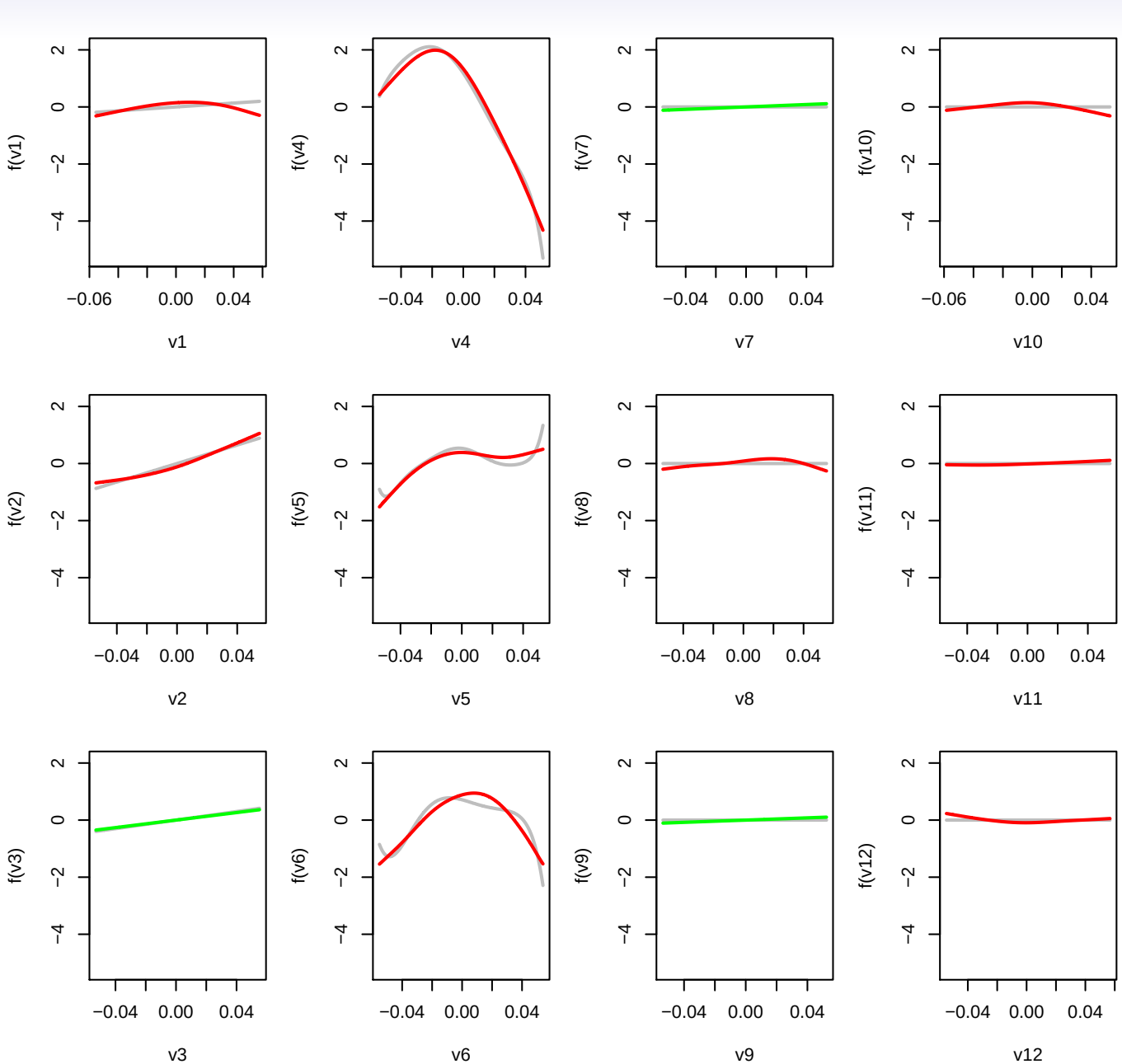

Step= 36 lambda = 4.68
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

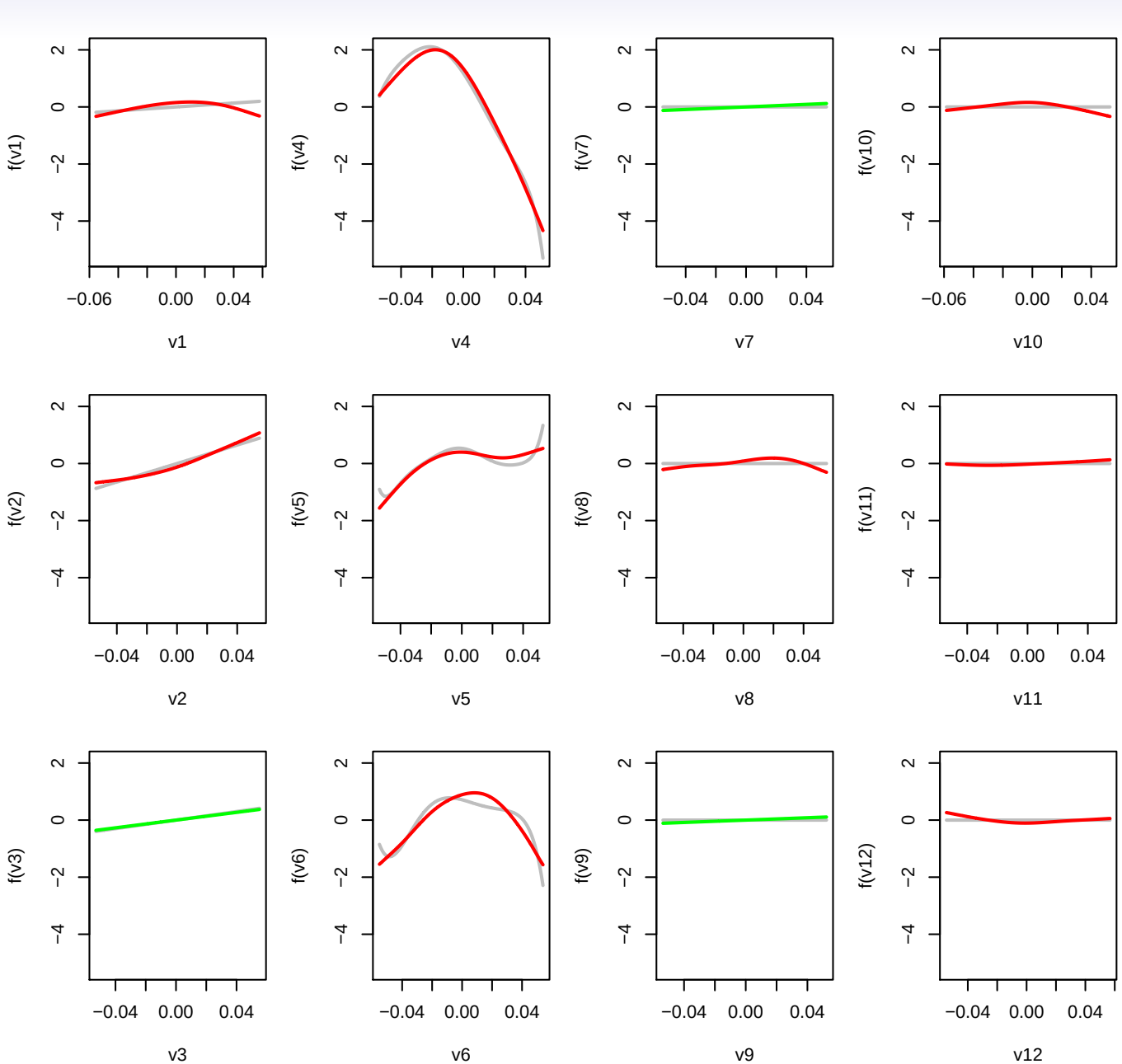

Step= 37 lambda = 4.26
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

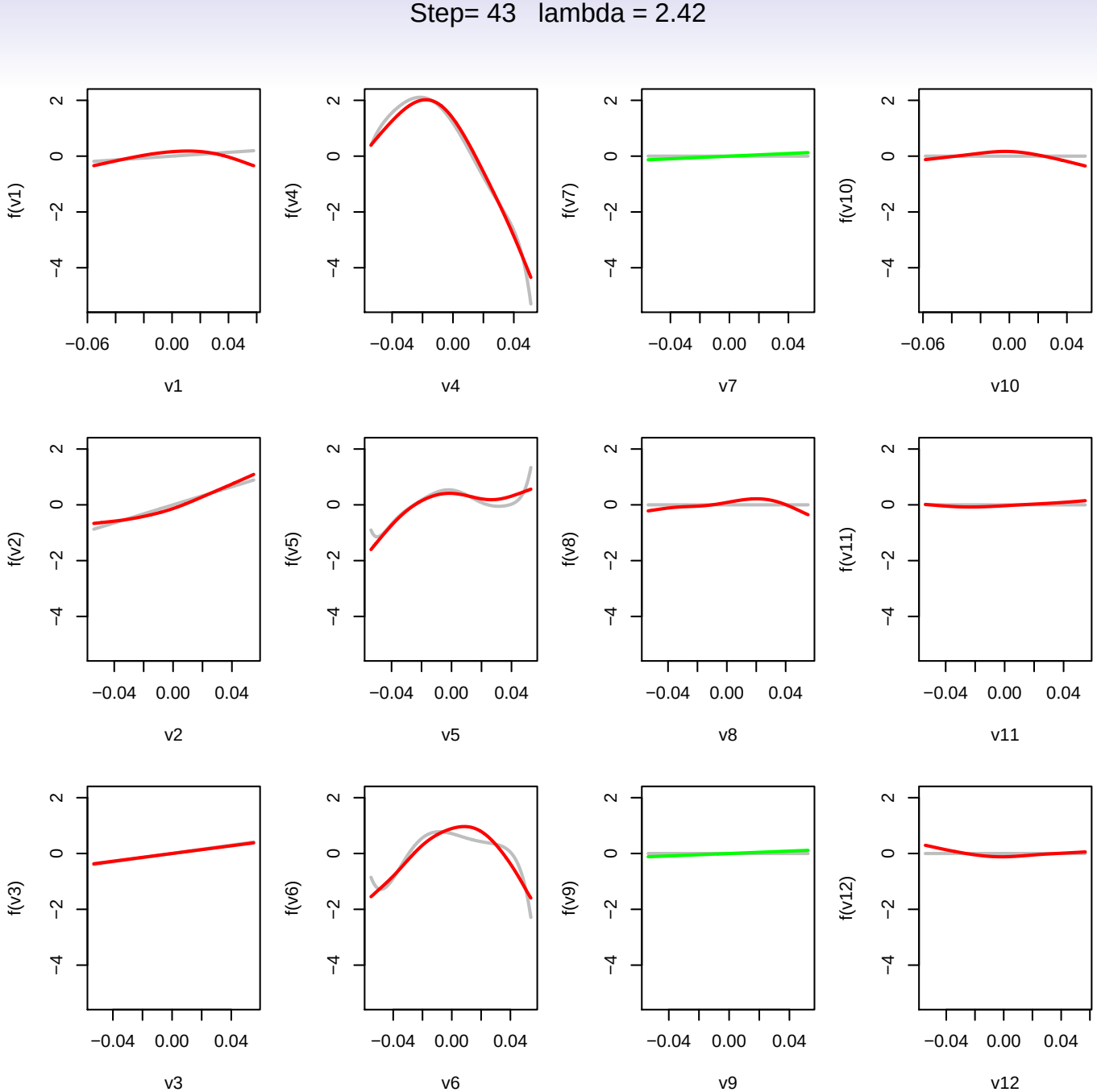

Step= 38 lambda = 3.87
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

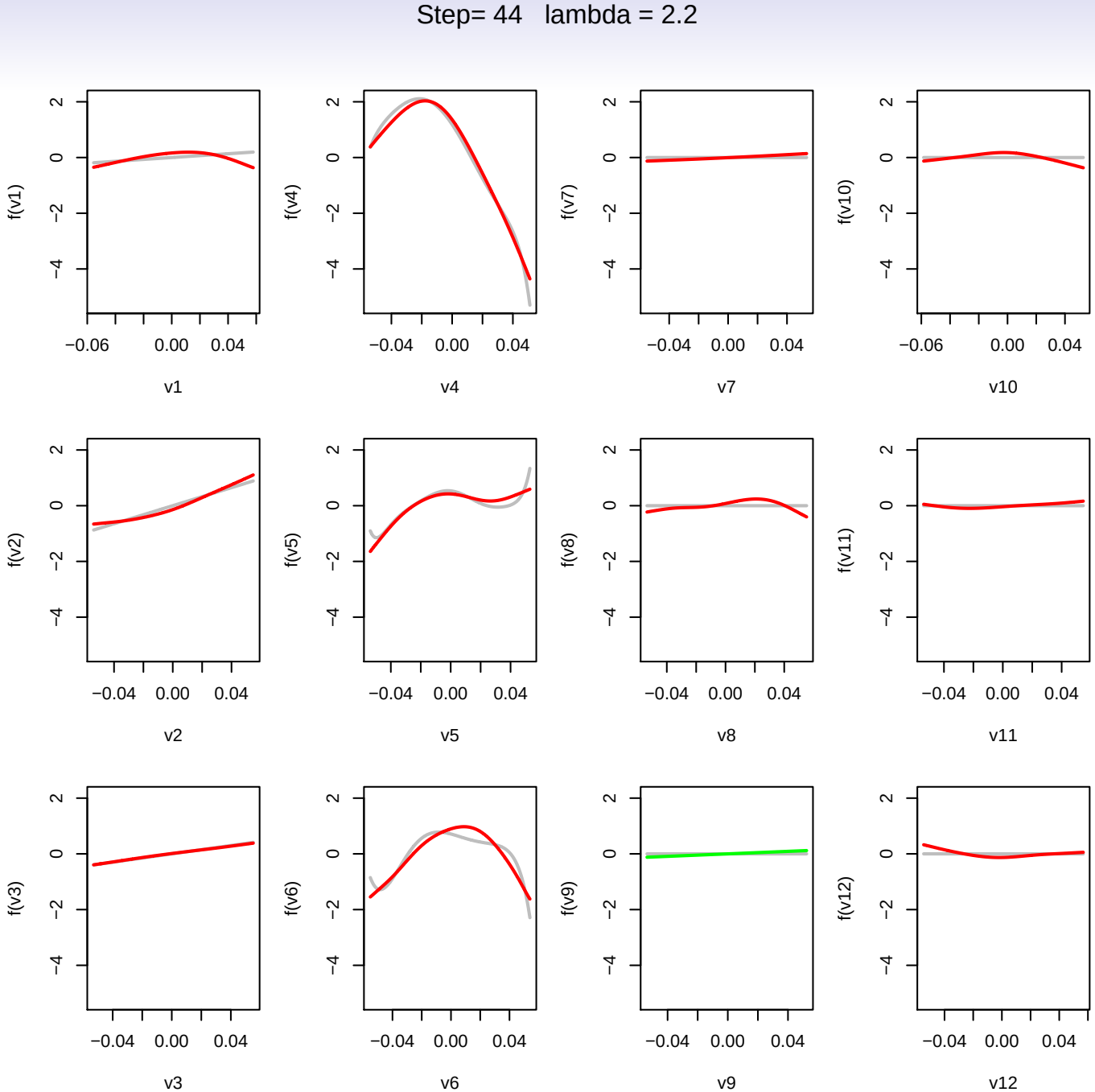

Step= 39 lambda = 3.53
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

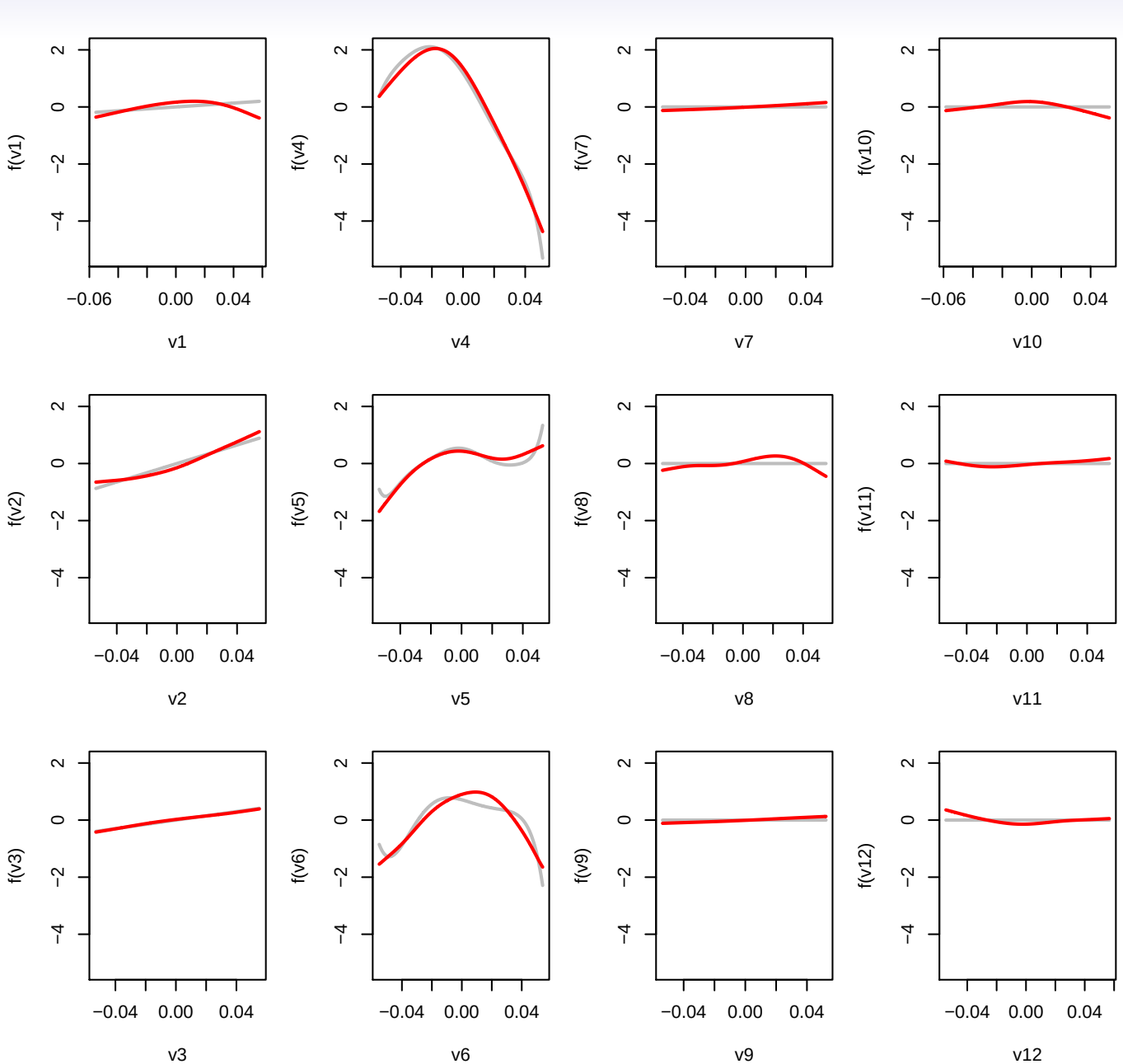

Step= 40 lambda = 3.21
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

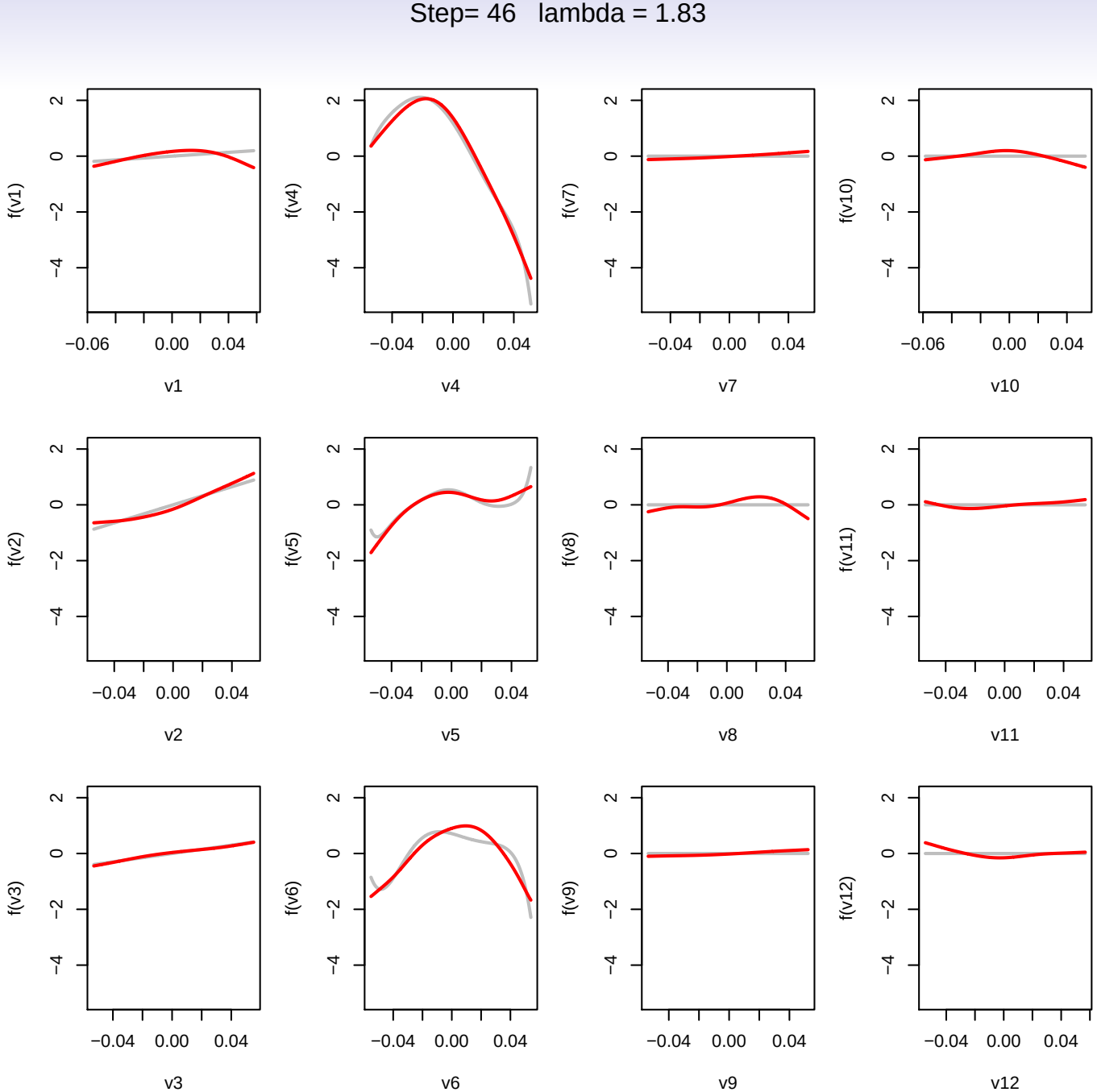

Step= 41 lambda = 2.92
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

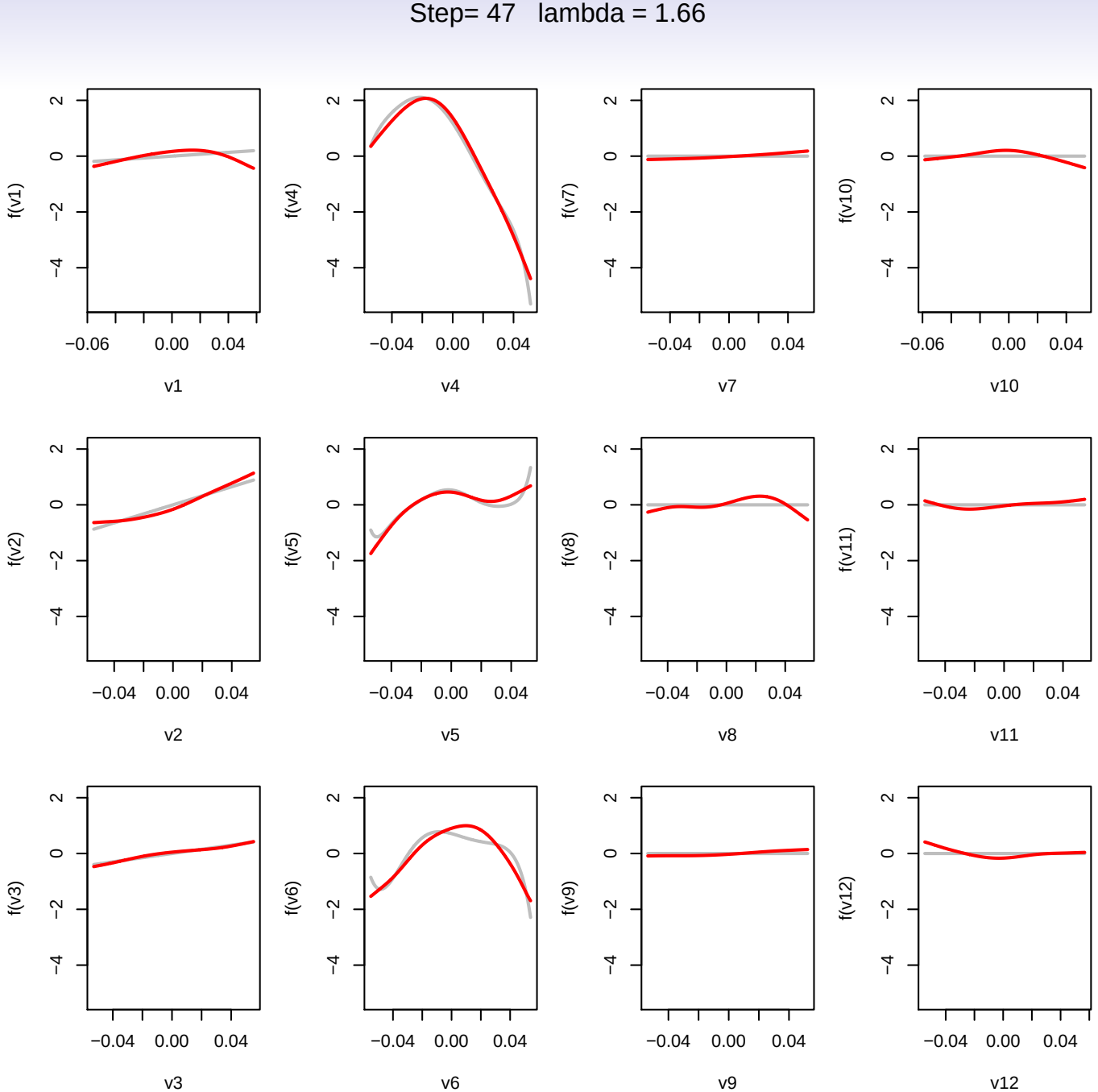

Step= 42 lambda = 2.66
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

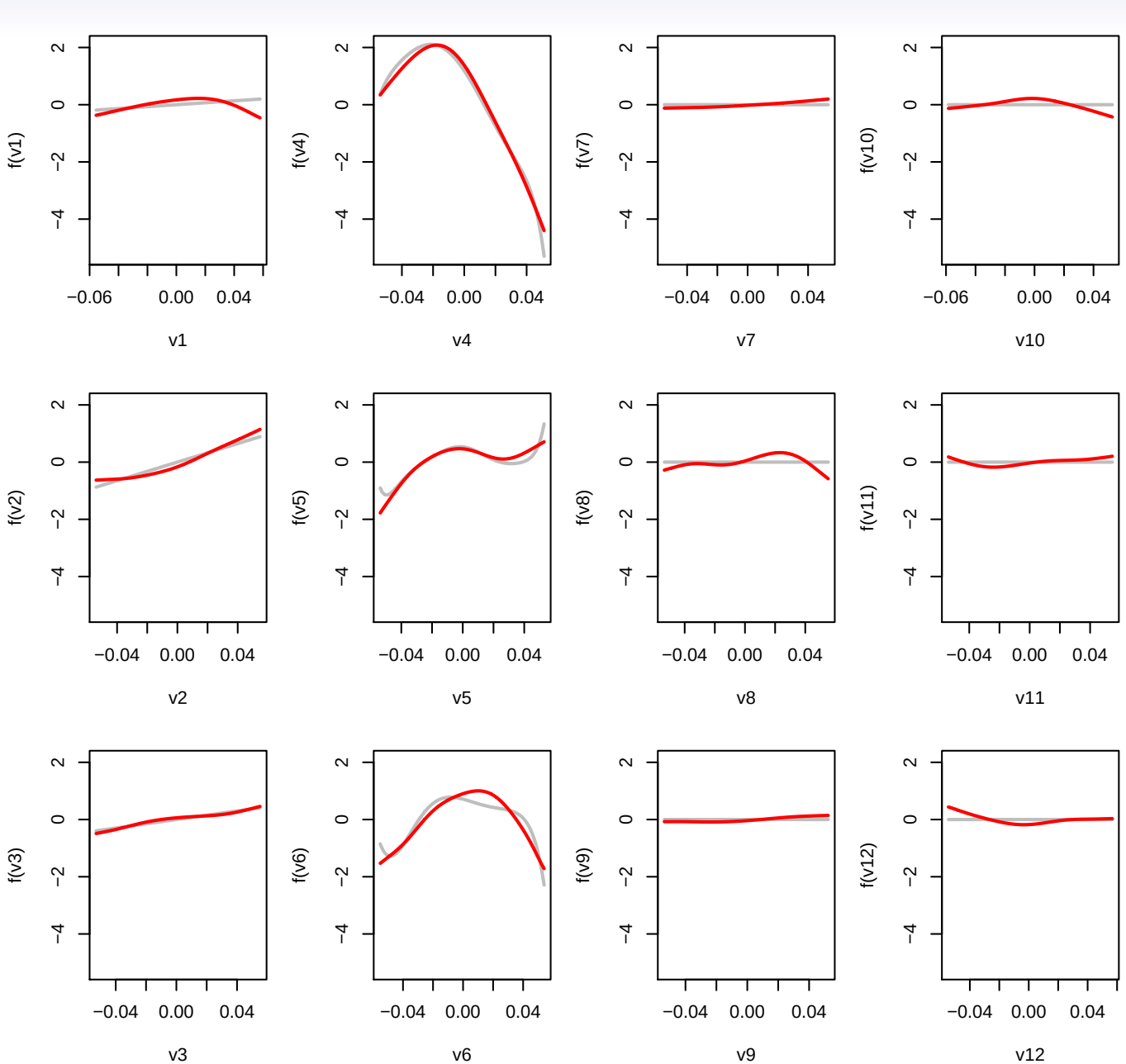

Step= 43 lambda = 2.42
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

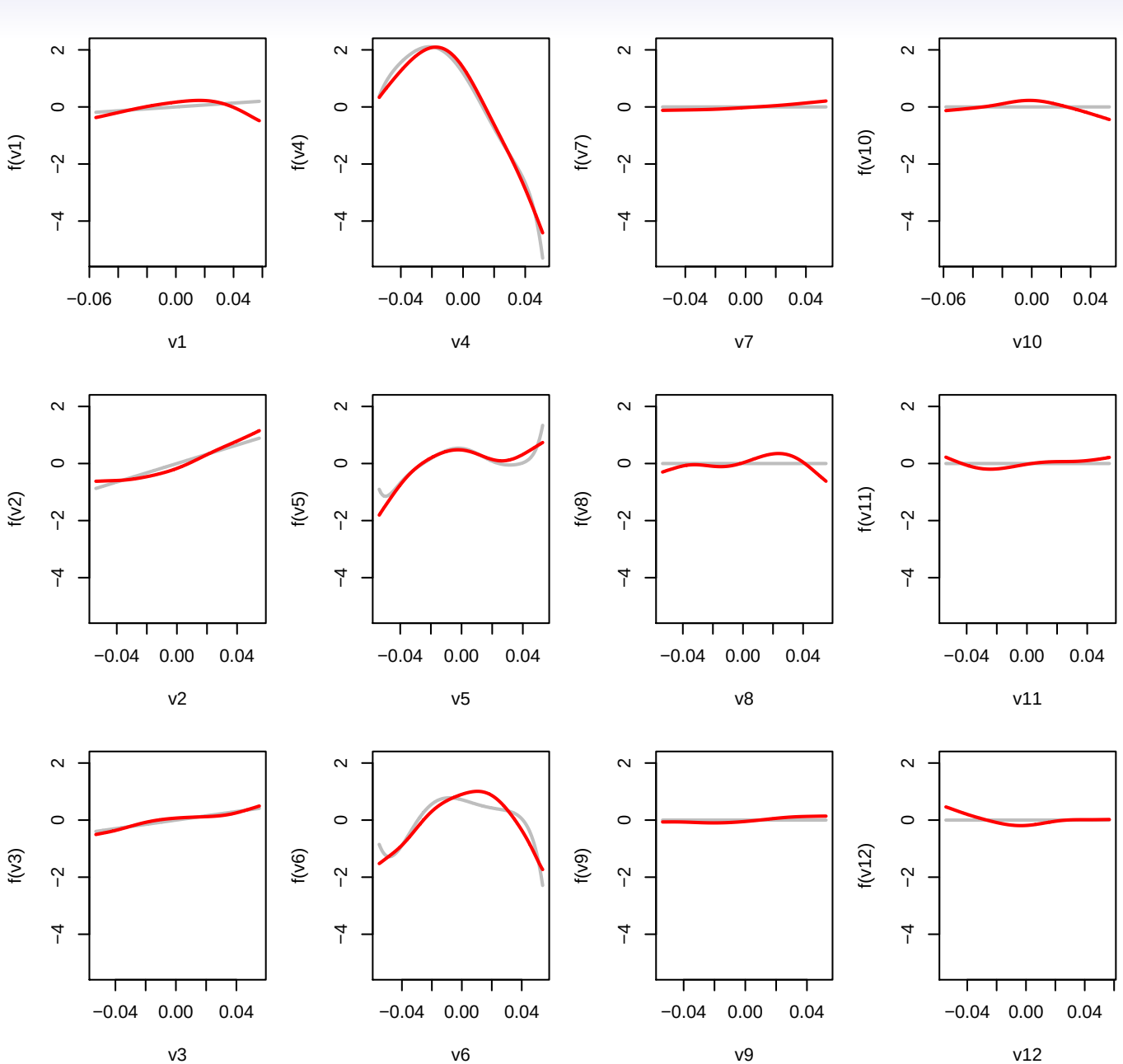

Step= 44 lambda = 2.2
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

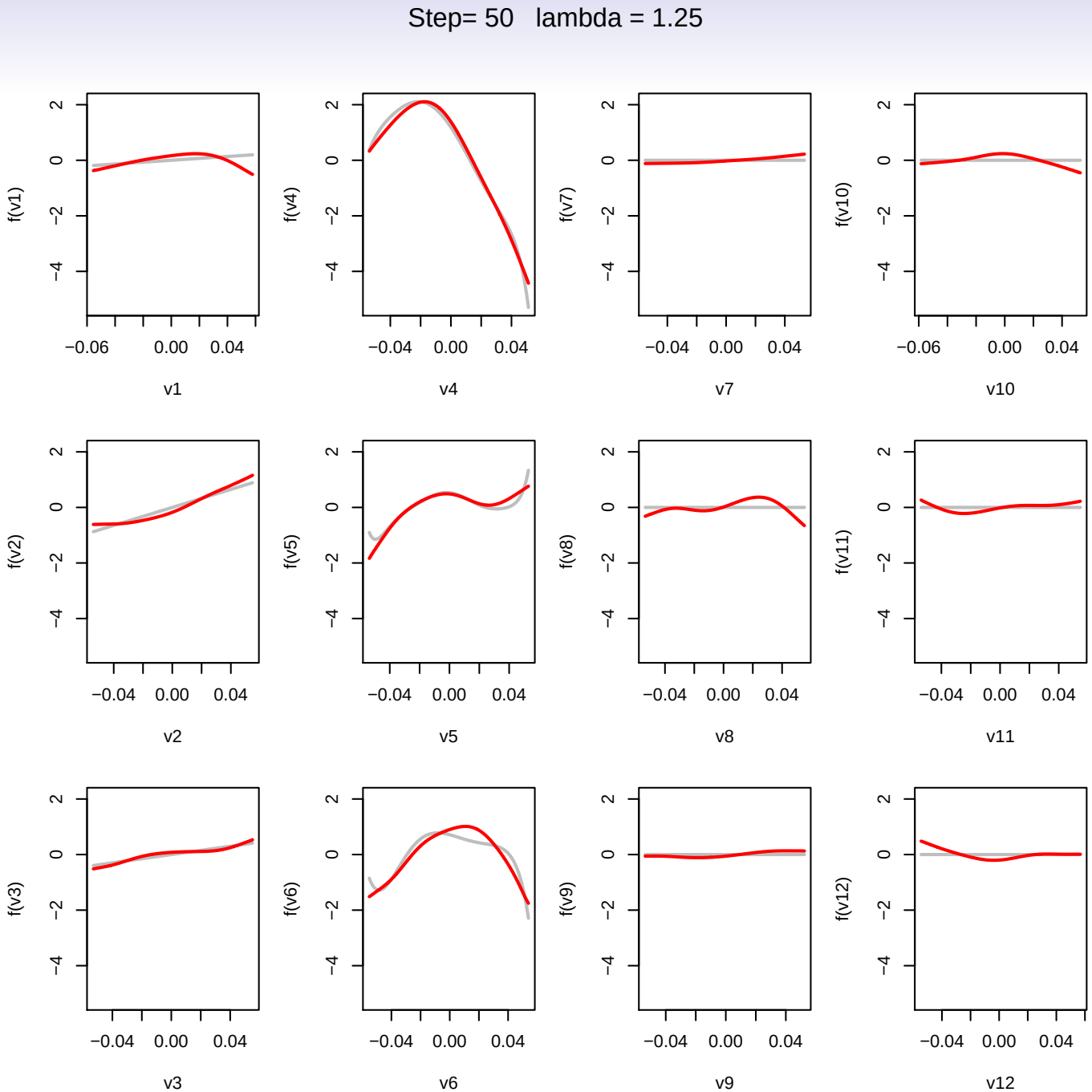

Step= 45 lambda = 2.01
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12


Step= 46 lambda = 1.83
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12


Step= 47 lambda = 1.66
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12


Step= 48 lambda = 1.51
f(v1)
f(v4)
f(v7)
f(v10)
f(v2)
f(v5)
f(v8)
f(v11)
f(v3)
f(v6)
f(v9)
f(v12)
v1
v4
v7
v10
v2
v5
v8
v11
v3
v6
v9
v12


Step= 49 lambda = 1.38
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12


Step= 50 lambda = 1.25
f(v1)
v1
f(v4)
v4
f(v7)
v7
f(v10)
v10
f(v2)
v2
f(v5)
v5
f(v8)
v8
f(v11)
v11
f(v3)
v3
f(v6)
v6
f(v9)
v9
f(v12)
v12

Thank you for your attention.

Thank you to Angela Montanari for inviting us, and for her generous support.