arp.gee
(version 0.1.0)

Affected Relative Pair Linkage Mapping With Generalized Estimating Equations

Jason P. Sinnwell and Daniel J. Schaid

Mayo Clinic, Division of Biostatistics
Rochester MN USA

April 6, 2005

Contents

1 Brief Description

The *arp.gee* package makes use of identical-by-descent (IBD) data to simultaneously estimate a trait-locus position and its genetic effects for affected relative pairs (ARPs) using generalized estimation equations (GEE) [1]. Two models are available for this estimation. One model allows a different trait-locus effect for each ARP type. The other model constrains the trait-locus effects according to the marginal effect of a single susceptibility locus. The function *arp.ibd* is the main function of this package, which carries out the estimation. This function considers different starting values for the trait-locus position, and chooses that position that gives the best goodness-of-fit. Functions are provided that plot the mean IBD sharing values and the fitted values from the model, provide a summary of the fit of a model, and that test the adequacy of the fit of the constrained model, relative to the unconstrained model.

2 Operating System and Installation

The *arp.gee* version 0.1.0 package is written for both S-PLUS (version 6.2) and R (version 2.0.1) for Unix. It is available on the Comprehensive R Archive Network (CRAN) where packages are made available for additional systems. Installation procedures for S-PLUS and R systems will vary; the Unix installations are explained in the *README.arp.gee* text file, located at the top level of the *arp.gee* directory. The procedures for running analyses are the same for S-PLUS and R, as illustrated in the instructions in this document.

3 Getting Started

The *arp.gee* package contains a data set with IBD data, *example.share*, that is used in the examples in this document. These examples also appear in a demo file located in the *demo* directory and in the help files. The user can import IBD data from genetic linkage software, either Merlin or Genehunter. Because this takes a number of steps, these steps will be illustrated.

For users new to the S-PLUS or R environments, note the following basic concepts. In the following examples, a user enters the indented text following the prompt ">", and the output results follow. Later, when executing a function in the session, the general syntax will appear like *'myresult <- myfunction(x)'* where the results of *myfunction*, operating on *x*, are saved in *myresult*. Then a user may print *myresult* or make use of it in a calculation.

To begin, if you have not done so, load the *arp.gee* package. If the *arp.gee* package is installed for global use, load the library as illustrated below. If installed as a local library, specify its location using the *lib.loc* parameter as shown in comments(##), specifying the path where the library was installed.

```
## If local library use:
        ## library(arp.gee, lib.loc="/full/local/path/")
```

```
> library(arp.gee)
```

3.1 IBD Data from Genetic Software Packages

To use the *arp.ibd* function, we first need to create a data structure that contains the IBD data. This special data structure has a certain type of class, called *ibd.share*. The IBD data must be calculated outside the S-PLUS or R session, so we assume that a file of IBD probabilities has been created by using Genehunter or Merlin. For this example, we use results from Genehunter. First, copy all files from the **arp.gee/exec/** directory that start with *'gh.example'* to the working directory for this session. Two of these files are *LINKAGE* format used as input files for Genehunter:

- *gh.example.pre:* the data file that includes the pedigree structures and genetic markers (pre-makeped linkage format)
- *gh.example.par:* the file with penetrance, allele frequencies, and genetic map

The data for the example in this document were made using a script file, supplied as *gh.example.in* within the *exec* directory. It contains the following lines:

```
photo example.out
ps off
count rec on
haplotype off
map function kosambi
load markers gh.example.par
scan pedigrees gh.example.pre
dump ibd
gh.example.ibd.out
quit
```

Genehunter created *gh.example.ibd.out*. This file name, along with the file name *gh.example.pre*, are input to the function *ibd.share.genehunter*. This function then creates a new *ibd.share* data object, which we call *example.share* for this illustration. This step is illustrated below.

```
> example.share <- ibd.share.genehunter(ibd.file = "gh.example.ibd.out",
+     pre.file = "gh.example.pre", min.pairs = 20)
```

The parameter *min.pairs* is used as the minimum count for an ARP type to become part of the *ibd.share* object. The user can apply this restriction because the methods of *arp.gee* may not be reliable for ARP types with small counts, particularly when attempting to estimate effects specific to each type of ARP.

The *ibd.share.genehunter* function categorizes ARPs into their types. First, the IBD probabilities under the null hypothsesis of no linkage (the prior IBD's from Genehunter) are compared to

their expected values for different types of relative pairs. However, this is not completely accurate, so an additional function, *relpair.type*, is used to evaluate the types of relative pairs contained in the *gh.example.pre* file. This latter function does some simple checks, and the results from this function are compared with those from Genehunter's priors for determining the final category for ARP. Because these files can be large, it can take a fair amount of time for *ibd.share.genehunter* to create the desired *ibd.share* object. So, it may be best to submit the above command as a batch process.

If for some reason, you cannot load the *example.share* object using the above steps, it can be loaded into your session using the following command.

```
> setupData(example.share)
```

3.2 The *ibd.share* Object

Below, we look at the names of the components in *example.share*.

```
> names(example.share)
```

```
[1] "smat" "ped"  "per1" "per2" "type" "pos"
```

The components of *example.share* are defined as follows:

- **smat:** Matrix of estimated IBD sharing values, arranged by ARP type (rows) and chromosome position (columns). Rows are sorted by ARP type and then pedigree (ped).
- **ped:** Pedigree code for all ARPs, elements correspond to rows in *smat*. Many relative pairs from a pedigree may be included, but they don't appear together in *smat* due to the sorting
- **per1:** ID code for one of the affected relatives in the pair; the codes are unique within pedigrees
- **per2:** ID code for the second of the affected relatives
- **type:** ARP type, as a factor, of all ARPs in the order they are stored in *smat*
- **pos:** Chromosome position, same as column names of *smat*

3.2.1 Summary of an *ibd.share* Object

The *summary* function provides a summary of the data in an *ibd.share* object.

```
> summary(example.share)
```

```
Counts of ARP Types

 FS  HS  FC  GP  AV
206   0 112   0  26

Number of positions :  61

Number of pedigrees :  82

         Dim of smat:  344 61
```

3.2.2 Plot an *ibd.share* Object

To view the mean IBD sharing for the different ARP types, use the *plot* command. The results from plot are illustrated in Figure 1, showing the mean IBD sharing by chromosome position for each ARP type. This gives a sense of the chromosome position of IBD sharing peaks.

4 Standard Use of *arp.ibd*

In this section we will execute *arp.ibd* using the default settings. We demonstrate the basic difference between the constrained and unconstrained models. Then we perform a test of the null hypothesis that the constrained model gives an adequate fit (i.e., the marginal trait-locus effect is the same for all ARP types).

4.1 The C Model (unconstrained)

First we estimate the trait-locus position (τ) while allowing a different trait-locus effect for each of the different ARP types. This is referred to as the "unconstrained model", because later we constrain the model so that the marginal effect of the trait-locus is the same for each of the types of ARPs. With the unconstrained model, we make use of C-coefficients, which represent the expected departure from random sharing at the trait-locus for the separate ARP types. The unconstrained model is called the C model within the *arp.ibd* function. Below we fit the C model and the results are saved in *fit.c*. We can print *fit.c* by entering it alone on the command line (equivalently we could have used *print(fit.c)*).

```
> fit.c <- arp.ibd(example.share, model = "C")
> fit.c

============================================================
          Parameter estimates and 95 % confidence
                          intervals
============================================================
```

```
      estimate   lowerCI  upperCI
tau  73.877492 65.200202 82.55478
C-FS  0.157499  0.029161  0.28584
C-FC  0.047851 -0.048448  0.14415
C-AV -0.053164 -0.294388  0.18806

============================================================
          Lambdas transformed from estimated Cs
============================================================

           lambda lowerCI upperCI
lambda-FS 1.45985 1.06193  2.3347
lambda-FC 1.27259 0.75729  1.9517
lambda-AV 0.80778 0.25883  2.2057
```

The first table shows estimates and confidence intervals for the C-coefficients, and *tau*, which is the estimated trait-locus position (τ). When we print a C model, we also see the C-coefficients translated to corresponding λ values, which are explained in section 4.3.

We can also see an extended summary of *fit.c* using the *summary* function.

```
> summary(fit.c)

============================================================
               Details per starting position
============================================================

  tau.init  tau.est      C.FS       C.FC        C.AV
1   31.987 73.87749 0.1574995 0.04785057 -0.05316397
2   51.685 73.87749 0.1574995 0.04785057 -0.05316398
3   76.294 73.87752 0.1574995 0.04785063 -0.05316386
   gof.model
1 0.01482689
2 0.01482689
3 0.01482689

 Number of pedigrees: 82
 epsilon:  1
 model:  C
 Converged after   24  N-R iterations.

============================================================
```

```
        Number of pairs for each relative pair type.
============================================================

         FS  FC AV
n.pairs 206 112 26

============================================================
          Parameter estimates and 95 % confidence
                        intervals
============================================================

      estimate   lowerCI  upperCI
tau  73.877492 65.200202 82.55478
C-FS  0.157499  0.029161  0.28584
C-FC  0.047851 -0.048448  0.14415
C-AV -0.053164 -0.294388  0.18806

============================================================
           Lambdas transformed from estimated Cs
============================================================

           lambda lowerCI upperCI
lambda-FS 1.45985 1.06193  2.3347
lambda-FC 1.27259 0.75729  1.9517
lambda-AV 0.80778 0.25883  2.2057
```

In the summary, the first table contains for each starting position of τ (*tau.init*), the parameter estimates and a goodness-of-fit measure (*gof.model*). Next are details of the Newton-Raphson (N-R) estimation process, then counts per ARP type in the dataset. The final portion is the same information as in the *print* output.

4.2 Plot the C Model Fit

To view how the fitted values of a model compare with the mean sharing, simply plot the *arp.ibd* result together with *example.share* as with the following command.
The plot is presented in Figure 2.

4.3 The λ Model (constrained)

Under the λ model, the C-coefficients are constrained to be a function of a single parameter λ (*lambda* in the syntax). This λ represents the ratio of risk for a relative who shares one trait allele

IBD with an affected person to the risk for a relative who shares no alleles IBD. Here we fit the model with default parameters for the λ model, and print the result.

```
> fit.lambda <- arp.ibd(example.share, model = "lambda")
> print(fit.lambda)

============================================================
          Parameter estimates and 95 % confidence
                          intervals
============================================================

       estimate lowerCI upperCI
tau     78.4135 68.9707  87.856
lambda   1.3795  1.0031   1.756
```

Since the λ model has only a single parameter that represents the trait-locus effect for all types of ARPs, the printed output from the λ model is just the table wit the two parameter estimates with confidence intervals.

4.4 Plot the λ Model Fit

The plot function can be used to view the fit of a model, except this time we show the plot without passing *example.share* to the plot function. The results are shown in Figure 3.

In this example, we only see the fitted lines, where the peak shows the estimated trait-locus position. The plot would be more informative for assessing the fit of the model if we also pass the *ibd.share* object, as in Figure 2. We see that the fitted peaks show the estimated trait-locus position is about the same as for the *C* model. However, we can also see by the shape of the peaks that the constrained model kept the effects for all three ARPs the same. This was not the case for *fit.c*, most noticeably in the plots for *type=AV*.

4.5 Test the Constrained Model

We can evaluate the fit of the constrained model by contrasting it with the unconstrained model, using a score statistic. This statistic tests whether the marginal trait-locus effect, scaled according to ARP, is the same across all types of ARPs. To test whether the unconstrained λ's from the different types of ARPs are significantly different from each other, we need the *ibd.share* object (*example.share*) and the results from fitting the constrained λ model (*fit.lambda*). These two objects are passed to the function *lambda.equal.arp*, as illustrated below. The result is a χ^2 statistic with the given degrees of freedom and p-value.

```
> stat.equal <- lambda.equal.arp(example.share,
+     fit.lambda)
> unlist(stat.equal)
```

```
     stat        df      pval
1.0818223 2.0000000 0.5822175
```

5 Additional Functionality

Sometimes the estimation may not converge under the default settings. In this section we explore some of the issues that may arise. The *arp.ibd* function allows the user to pass starting values for the parameters (τ, and either the C-coefficients or a λ). The default settings in *arp.ibd* select reasonable starting values, all based on IBD sharing values shown in Figure 1. However, we demonstrate, by fitting separate models to each ARP type, that the estimation process may need to be altered to achieve convergence.

5.1 Full-Sibs Subset

The Full-Sibs ARP type, as evident in Figure 2, appears to be best fit to the C model. Here, we fit the model-C to the Full-Sibs alone, but first we need to create an *ibd.share* object for the subset of Full-Sibs. To do this, we copy the *ibd.share* object (*example.share*), and then subset the elements in this new object to the desired ARP type. Full-Sibs are coded as "FS", so we create a logical vector, *fs*, having values of *TRUE* or *FALSE* according to whether an ARP is a Full-Sib or not. The example below demonstrates how we create an *ibd.share* object for Full-Sibs. Then we can run *arp.ibd* on the new *ibd.share* object.

```
> example.fs <- example.share
> fs <- example.share$type == "FS"
> example.fs$smat <- example.fs$smat[fs, ]
> example.fs$ped <- example.fs$ped[fs]
> example.fs$type <- example.fs$type[fs]
> save.fs <- arp.ibd(example.fs, model = "C")
> save.fs

 ============================================================
          Parameter estimates and 95 % confidence
                        intervals
 ============================================================

      estimate  lowerCI  upperCI
 tau  72.85727 63.51817 82.19636
 C-FS  0.15648  0.02834  0.28463

 ============================================================
           Lambdas transformed from estimated Cs
```

```
============================================================

           lambda lowerCI upperCI
lambda-FS 1.4555  1.0601  2.3215
```

5.2 First-Cousins: Multiple Peaks

We apply the same subsetting technique for First-Cousins. For this group, we see in the middle plot in Figure 1 that there are multiple peaks of mean IBD sharing, so *arp.ibd* may give more than one estimate of a trait-locus position. Here we specify three starting values for τ by using *tau.init*. Then, by using the summary function, we can compare the goodness-of-fit values from each of the starting τ positions which converged.

```
> example.fc <- example.share
> fc <- example.share$type == "FC"
> example.fc$smat <- example.fc$smat[fc, ]
> example.fc$ped <- example.fc$ped[fc]
> example.fc$type <- example.fc$type[fc]
> save.fc <- arp.ibd(example.fc, model = "C", tau.init = c(30,
+     60, 90))
> summary(save.fc)

============================================================
               Details per starting position
============================================================

  tau.init  tau.est       C.FC   gof.model
1       30 39.39064 0.03696672 0.002254816
2       60 93.77981 0.07012804 0.007038550
3       90 93.77982 0.07012804 0.007038550

 Number of pedigrees: 34
 epsilon:  1
 model:  C
 Converged after   23  N-R iterations.

============================================================
        Number of pairs for each relative pair type.
============================================================

         FC
n.pairs 112
```

```
============================================================
          Parameter estimates and 95 % confidence
                          intervals
============================================================

      estimate   lowerCI   upperCI
tau  93.779818 77.167885 110.39175
C-FC  0.070128 -0.012917   0.15317

============================================================
           Lambdas transformed from estimated Cs
============================================================

           lambda lowerCI upperCI
lambda-FC 1.4126 0.93228  2.0266
```

5.3 Avuncular Pairs: Non-Convergence

Finally we apply the subsetting procedure to Avuncular Pairs (coded as 'AV'). Recall the plot for *example.share* (Figure 1), where the average IBD sharing values for Avuncular Pairs showed a low IBD-sharing region in the middle, and high sharing near the right end. The *arp.ibd* function chooses default starting values where the sharing is high, and in this case, chooses a starting value near the right end. We see below that the estimation does not converge.

```
> example.av <- example.share
> av <- example.share$type == "AV"
> example.av$smat <- example.av$smat[av, ]
> example.av$ped <- example.av$ped[av]
> example.av$type <- example.av$type[av]
> save.av <- arp.ibd(example.av, model = "C")
> save.av

 Convergence not met

 ============================================================
                Parameter estimates and scores
 ============================================================

       param u.scores
1 90.231000 0.073136
2  0.014416 0.015853
```

```
No. iterations =  1
```

To attempt convergence, we now specify alternative starting values for the trait-locus position (τ) by the parameter *tau.init*.

```
> save.av.init <- arp.ibd(example.av, model = "C",
+     tau.init = c(30, 90, 60))
> save.av.init

============================================================
          Parameter estimates and 95 % confidence
                         intervals
============================================================

      estimate  lowerCI  upperCI
tau  51.799431 13.92347 89.67540
C-AV -0.077358 -0.34331  0.18860

============================================================
           Lambdas transformed from estimated Cs
============================================================

            lambda lowerCI upperCI
lambda-AV 0.73203  0.1858  2.2113
```

By viewing the *summary.df* data frame, we see that no information was kept for the estimation when *tau.init* was 90, and that starting at 30 and 60 both converged to the same location, $\tau = 51.8$.

```
> save.av.init$summary.df

  tau.init  tau.est        C.AV   gof.model
1       30 51.79943 -0.07735812 0.008752643
2       60 51.79939 -0.07735816 0.008752643
```

By viewing the fit using the plot function, we see in Figure 4 that the fit is actually for a valley, suggesting that modeling a trait-locus position for AV pairs in this example may be misleading.

6 License and Warranty

Daniel J. Schaid, Ph.D.
Division of Biostatistics
Harwick Building - Room 775
Mayo Clinic
200 First St., SW
Rochester, MN 55905
phone: 507-284-0639
fax: 507-284-9542
email: schaid@mayo.edu

7 Acknowledgements

This research was supported by United States Public Health Services, National Institutes of Health; Contract grant number GM67768.

References

[1] Schaid DJ, Sinnwell JP, Thibodeau SN (2005). Robust Multipoint Identical-by-Descent Mapping for Affected Relative Pairs. *Am J Hum Genet* 76:128-138.

[2] Liang K-Y, Chiu Y-F, Beaty TH (2001). A Robust Identity-By-Descent Procedure Using affect Sib Pairs: Multipoint Mapping for Complex Diseases. *Hum Hered* 51:64-78.

```
> plot(example.share)
```

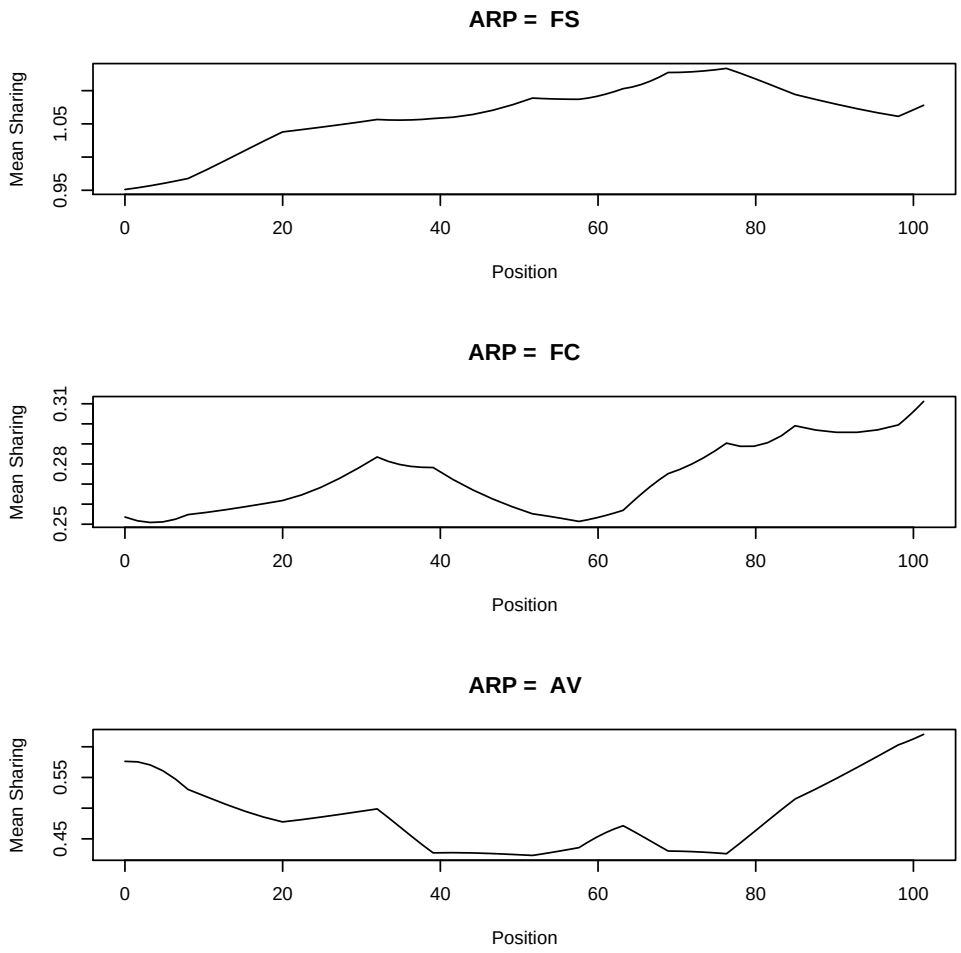

Figure 1: Mean IBD sharing by chromosome position for each ARP type

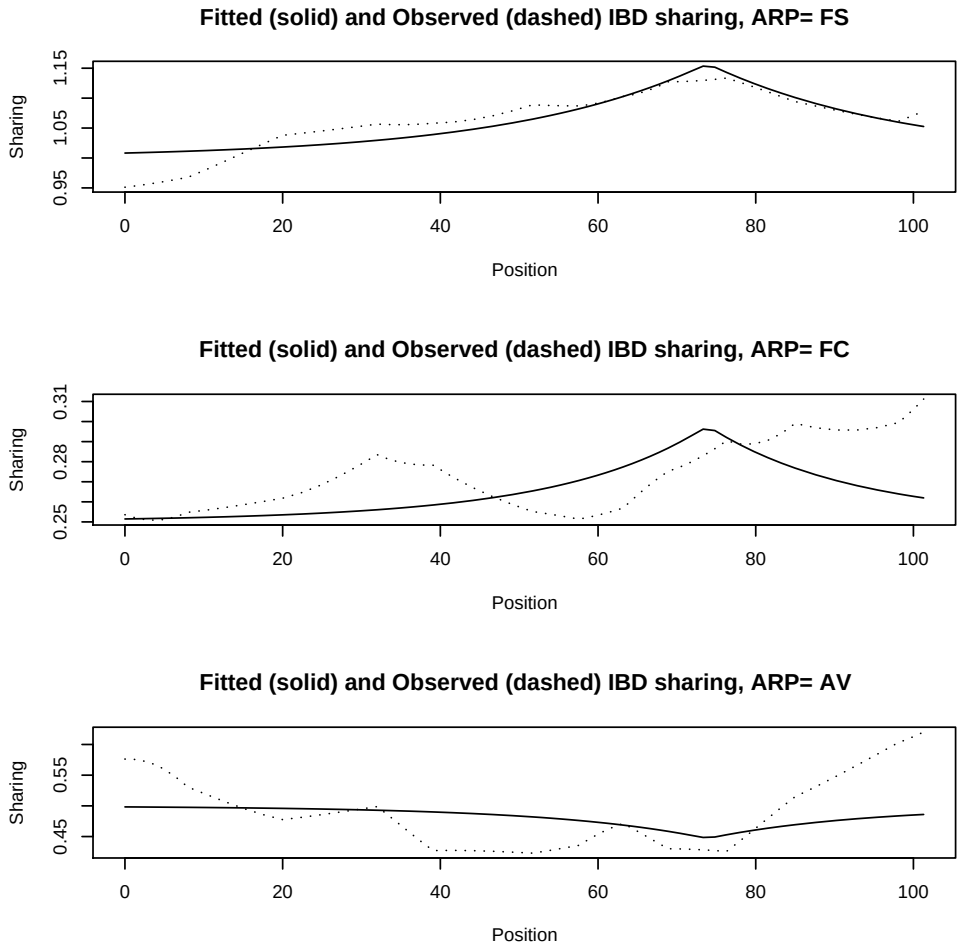

Figure 2: Plot of a *C* model *arp.ibd* object and the mean sharing data from *example.share*

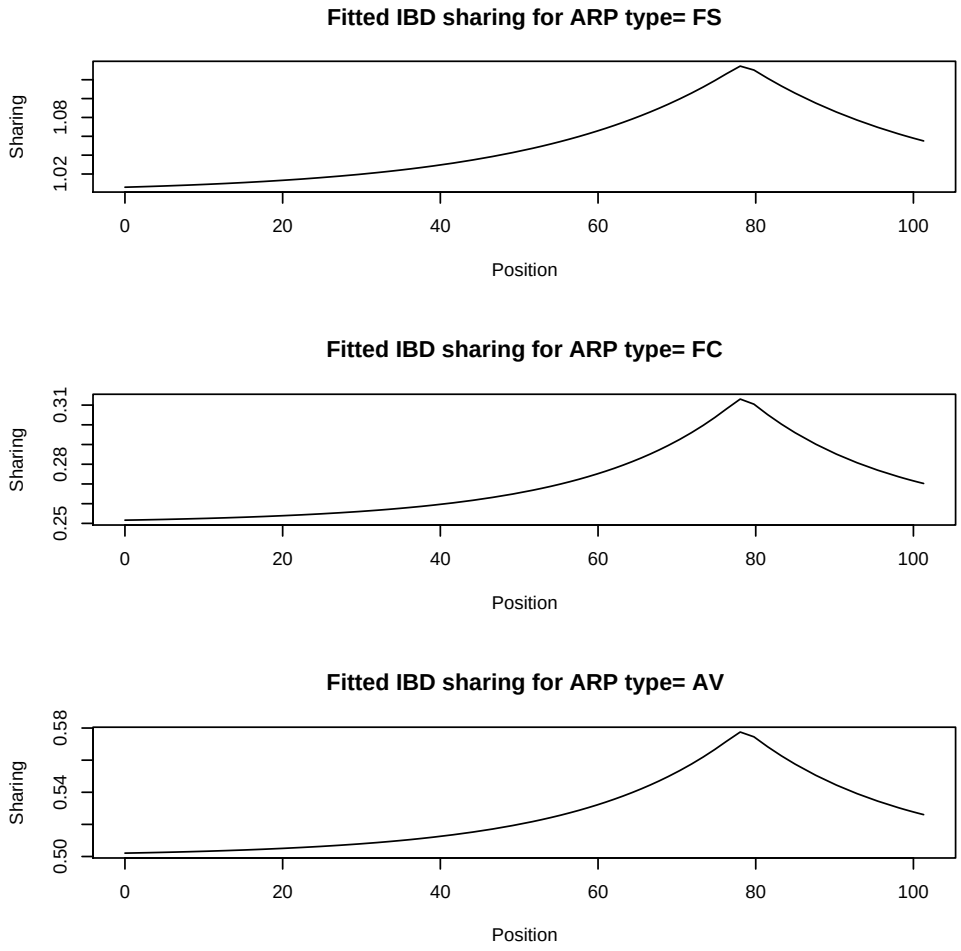

Figure 3: Plot a *lambda* model *arp.ibd* object, no *ibd.share* data

```
> plot(save.av.init, example.av)
```

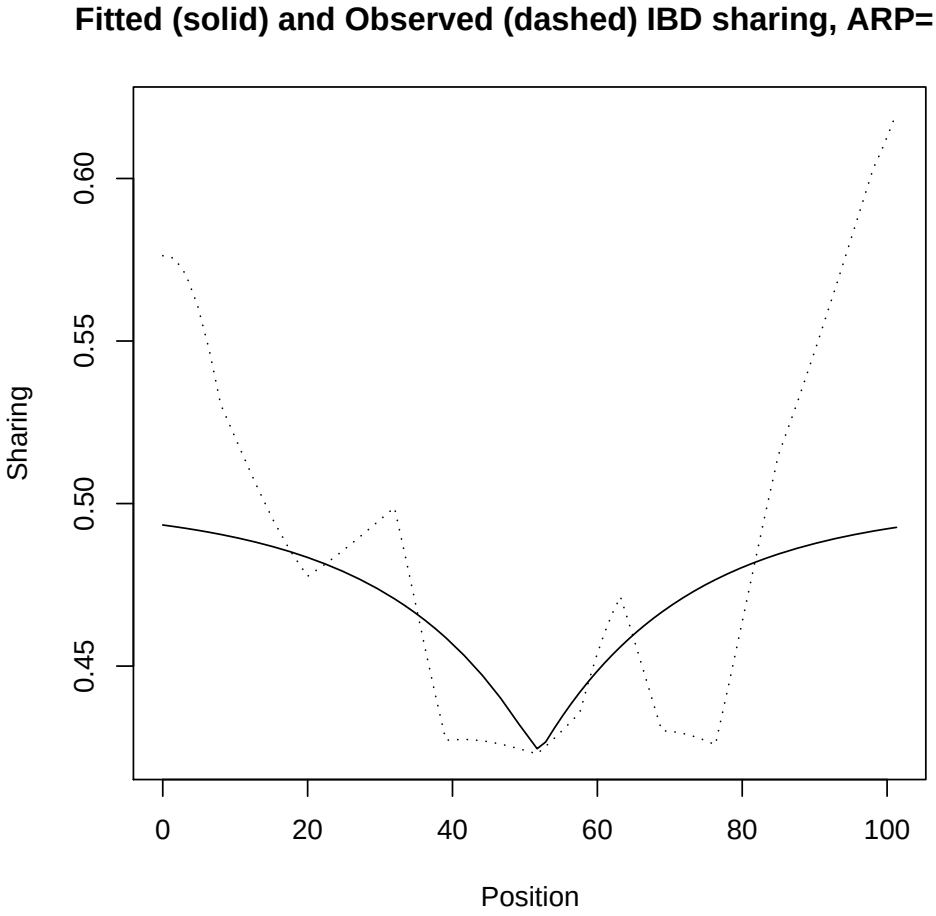

Figure 4: Plot *arp.ibd* fit for AV pairs and mean IBD sharing data from *example.av*