Maximum likelihood estimation and analysis with the `bbmle` package

Ben Bolker

April 25, 2010

Contents

Note: I have suppressed the continuation character (+) in the R examples throughout this document, as I find it easier to read/cut-and-paste where necessary.

The `bbmle` package, designed to simplify maximum likelihood estimation and analysis in R, extends and modifies the `mle` function and class in the `stats4` package that comes with R by default. `mle` is in turn a wrapper around the `optim` function in base R. The maximum-likelihood-estimation function and class in `bbmle` are both called `mle2`, to avoid confusion and conflict with the original functions in the `stats4` package. The major differences between `mle` and `mle2` are:

- `mle2` is slightly more robust, with additional warnings (e.g. if the Hessian can't be computed by finite differences, `mle2` returns a fit with a missing Hessian rather than stopping with an error)
- `mle2` uses a `data` argument to allow different data to be passed to the negative log-likelihood function
- `mle2` has a formula interface like that of (e.g.) `gls` in the `nlme` package. For relatively simple models the formula for the maximum likelihood can be written in-line, rather than defining a negative log-likelihood function. The formula interface also simplifies fitting models with categorical variables. Models fitted using the formula interface also have applicable `predict` and `simulate` methods.

- `bbmle` defines `anova`, `AIC`, `AICc`, and `BIC` methods for `mle2` objects, as well as `AICtab`, `BICtab`, `AICctab` functions for producing summary tables of information criteria for a set of models.

Other packages with similar functionality (extending GLMs in various ways) are `aod` and `vgam` (on CRAN), `gnlr` and `gnlr3` in Jim Lindsey's `gnlm` package (http://popgen.unimaas.nl/~jlindsey/rcode.html).

1 Example

This example will use the classic data set on *Orobanche* germination from Crowder (1978) (you can also use `glm(...,family="quasibinomial")` or the `aod` package to analyze these data).

1.1 Test basic fit to simulated beta-binomial data

First, generate a single beta-binomially distributed set of points as a simple test.

Load the `emdbook` package to get functions for the beta-binomial distribution (density and random-deviate function — these functions are also available in Jim Lindsey's `rmutil` package).

```
> library(emdbook)
```

Generate random deviates from a random beta-binomial:

```
> set.seed(1001)
> x1 = rbetabinom(n=1000,prob=0.1,size=50,theta=10)
```

Load the package:

```
> library(bbmle)
```

Construct a simple negative log-likelihood function:

```
> mtmp <- function(prob,size,theta) {
   -sum(dbetabinom(x1,prob,size,theta,log=TRUE))
 }
```

Fit the model — use `data` to pass the `size` parameter (since it wasn't hard-coded in the `mtmp` function):

```
> (m0 <- mle2(mtmp,start=list(prob=0.2,theta=9),data=list(size=50)))

Call:
mle2(minuslogl = mtmp, start = list(prob = 0.2, theta = 9), data = list(size = 50))

Coefficients:
      prob      theta
 0.1030974 10.0758090

Log-likelihood: -2723.5
```

The `summary` method for `mle2` objects shows the parameters; approximate standard errors (based on quadratic approximation to the curvature at the maximum likelihood estimate); and a test of the parameter difference from zero based on this standard error and on an assumption of normality.

```
> summary(m0)

Maximum likelihood estimation

Call:
mle2(minuslogl = mtmp, start = list(prob = 0.2, theta = 9), data = list(size = 50))

Coefficients:
        Estimate Std. Error z value     Pr(z)
prob   0.1030974  0.0031626  32.599 < 2.2e-16 ***
theta 10.0758090  0.6213319  16.216 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

-2 log L: 5446.995
```

Construct the likelihood profile (you can apply `confint` directly to `m0`, but if you're going to work with the likelihood profile (e.g. plotting, or looking for confidence intervals at several different α values) then it is more efficient to compute the profile once):

```
> p0 <- profile(m0)
```

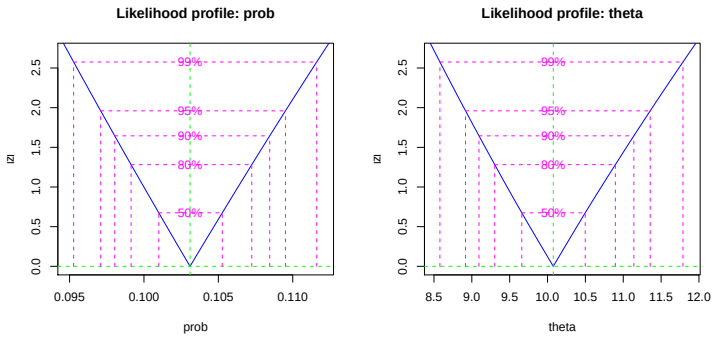
Compare the confidence interval estimates based on inverting a spline fit to the profile (the default); based on the quadratic approximation at the maximum likelihood estimate; and based on root-finding to find the exact point where the profile crosses the critical level.

```
> confint(p0)

           2.5 %     97.5 %
prob  0.09709228  0.1095103
theta 8.91708205 11.3559592

> confint(m0,method="quad")

           2.5 %     97.5 %
prob  0.09689875  0.1092961
theta 8.85802088 11.2935972

> confint(m0,method="uniroot")

           2.5 %     97.5 %
prob  0.09709185  0.1095099
theta 8.91691019 11.3559746
```

All three types of confidence limits are similar.

Plot the profiles:

```
> par(mfrow=c(1,2))
> plot(p0,plot.confstr=TRUE)
```



By default, the plot method for likelihood profiles displays the square root of the the deviance (twice the difference in negative log-likelihood), so it will be V-shaped for cases where the quadratic approximation works well (as in this case). (For a better visual estimate of whether the profile is quadratic, use the `absVal=FALSE` option to the `plot` method.)

You can also request confidence intervals calculated using `uniroot`, which may be more exact when the profile is not smooth enough to be modeled accurately by a spline. However, this method is also more sensitive to numeric problems.

Instead of defining an explicit function for `minuslogl`, we can also use the formula interface. The formula interface assumes that the density function given (1) has `x` as its first argument (if the distribution is multivariate, then `x` should be a matrix of observations) and (2) has a `log` argument that will return the log-probability or log-probability density if `log=TRUE`.

```
> m0f <- mle2(x1~dbetabinom(prob,size=50,theta),
              start=list(prob=0.2,theta=9))
```

It's convenient to use the formula interface to try out likelihood estimation on the transformed parameters:

```
> m0cf <- mle2(x1~dbetabinom(prob=plogis(lprob),size=50,theta=exp(ltheta)),
              start=list(lprob=0,ltheta=2))
> confint(m0cf,method="uniroot")

           2.5 %    97.5 %
lprob  -2.229963 -2.095757
ltheta  2.187950  2.429744

> confint(m0cf,method="spline")
```

```
           2.5 %    97.5 %
lprob  -2.229963 -2.095756
ltheta  2.187948  2.429742
```

In this case the answers from `uniroot` and `spline` (default) methods barely differ.

1.2 Using real data

Get data from Crowder (1978), as incorporated in the `aod` package:

```
> library(aod)

Package aod, version 1.1-33

> data(orob1)
```

Now construct a negative log-likelihood function that differentiates among groups:

```
> ML1 <- function(prob1,prob2,prob3,theta,x) {
   prob <- c(prob1,prob2,prob3)[as.numeric(x$dilution)]
   size <- x$n
   -sum(dbetabinom(x$y,prob,size,theta,log=TRUE))
 }
```

Results from Crowder (1978):

model	prob1	prob2	prob3	theta	sd.prob1	sd.prob2	sd.prob3	NLL
prop diffs	0.132	0.871	0.839	78.424	0.027	0.028	0.032	−34.991
full model								−34.829
homog model								−56.258

```
> (m1 <- mle2(ML1,start=list(prob1=0.5,prob2=0.5,prob3=0.5,theta=1),
     data=list(x=orob1)))
```

The result warns us that the optimization has not converged; we also don't match Crowder's results for θ exactly. We can fix this by setting `parscale` appropriately.

```
> (m2 <- mle2(ML1,start=as.list(coef(m1)),
           control=list(parscale=coef(m1)),
           data=list(x=orob1)))
```

Calculate likelihood profile (restrict the upper limit of θ, simply because it will make the picture below a little bit nicer):

```
> p2 <- profile(m2,prof.upper=c(Inf,Inf,Inf,theta=2000))
```

Get the curvature-based parameter standard deviations (which Crowder used rather than computing likelihood profiles):

```
> round(sqrt(diag(vcov(m2))),3)

 prob1  prob2  prob3  theta
 0.028  0.029  0.032 74.238
```

We are slightly off Crowder's numbers — rounding error?

Crowder also defines a variance (overdispersion) parameter $\sigma^2 = 1/(1+\theta)$.

```
> sqrt(1/(1+coef(m2)["theta"]))

    theta
0.1122089
```

Using the delta method (via the `deltavar` function in the `emdbook` package) to approximate the standard deviation of σ:

```
> sqrt(deltavar(sqrt(1/(1+theta)),meanval=coef(m2)["theta"],
          vars="theta",Sigma=vcov(m2)[4,4]))

[1] 0.05244185
```

Another way to fit in terms of σ rather than θ is to compute $\theta = 1/\sigma^2 - 1$ on the fly in a formula:

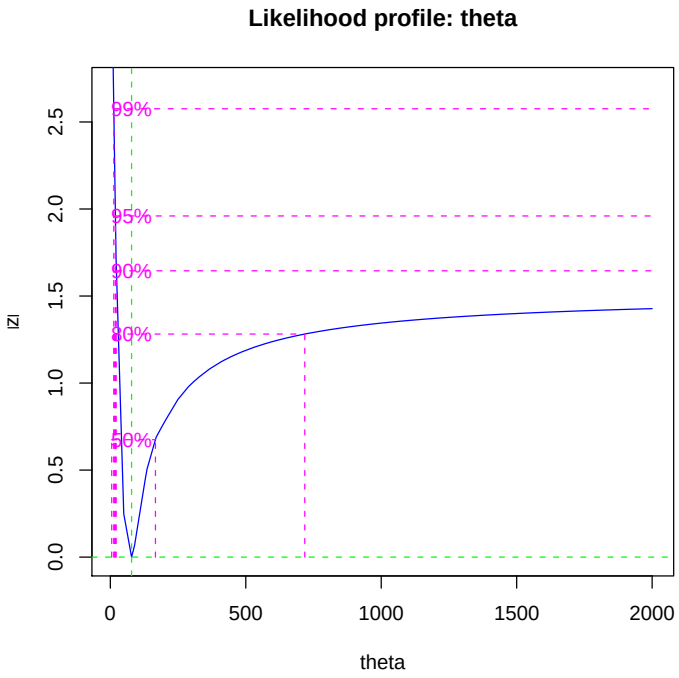
```
> m2b <- mle2(y~dbetabinom(prob,size=n,theta=1/sigma^2-1),
              data=orob1,
              parameters=list(prob~dilution,sigma~1),
              start=list(prob=0.5,sigma=0.1))
> round(sqrt(diag(vcov(m2b))),3)["sigma"]

sigma
0.052

> p2b <- profile(m2b,prob.lower=c(-Inf,-Inf,-Inf,0))
```

As might be expected since the standard deviation of σ is large, the quadratic approximation is poor:

```
> r1 <- rbind(confint(p2)["theta",],
              confint(m2,method="quad")["theta",])
> rownames(r1) <- c("spline","quad")
> r1

           2.5 %   97.5 %
spline  19.67166       NA
quad   -67.08083 223.9264
```

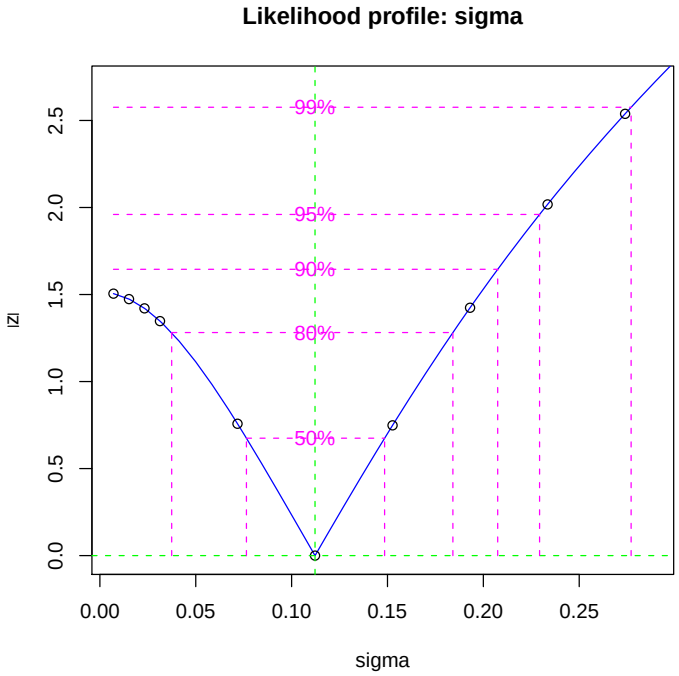
Plot the profile:

```
> plot(p2,which="theta",plot.confstr=TRUE)
```



What does the profile for σ look like?

```
> plot(p2b,which="sigma",plot.confstr=TRUE,
      show.points=TRUE)
```



Now fit a homogeneous model:

```
> ml0 <- function(prob,theta,x) {
   size <- x$n
   -sum(dbetabinom(x$y,prob,size,theta,log=TRUE))
 }
> m0 <- mle2(ml0,start=list(prob=0.5,theta=100),
           data=list(x=orob1))
```

The log-likelihood matches Crowder's result:

```
> logLik(m0)

'log Lik.' -56.25774 (df=2)
```

It's easier to use the formula interface to specify all three of the models fitted by Crowder (homogeneous, probabilities differing by group, probabilities and overdispersion differing by group):

```
> m0f <- mle2(y~dbetabinom(prob,size=n,theta),
             parameters=list(prob~1,theta~1),
             data=orob1,
             start=list(prob=0.5,theta=100))
> m2f <- mle2(y~dbetabinom(prob,size=n,theta),
```

```
             parameters=list(prob~dilution,theta~1),
             data=orob1,
             start=list(prob=0.5,theta=78.424))
> m3f <- mle2(y~dbetabinom(prob,size=n,theta),
             parameters=list(prob~dilution,theta~dilution),
             data=orob1,
             start=list(prob=0.5,theta=78.424))
```

anova runs a likelihood ratio test on nested models:

```
> anova(m0f,m2f,m3f)

Likelihood Ratio Tests
Model 1: m0f, y~dbetabinom(prob,size=n,theta): prob~1, theta~1
Model 2: m2f, y~dbetabinom(prob,size=n,theta): prob~dilution, theta~1
Model 3: m3f, y~dbetabinom(prob,size=n,theta): prob~dilution,
          theta~dilution
  Tot Df Deviance   Chisq Df Pr(>Chisq)
1      2  112.515
2      4   69.981 42.5341  2  5.805e-10 ***
3      6   69.981  0.0008  2     0.9996
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The various ICtab commands produce tables of information criteria, optionally sorted and with model weights.

```
> AICtab(m0f,m2f,m3f,weights=TRUE,delta=TRUE,sort=TRUE)

    dAIC df weight
m2f  0.0 4  0.881
m3f  4.0 6  0.119
m0f 38.5 2  <0.001

> BICtab(m0f,m2f,m3f,delta=TRUE,nobs=nrow(orob1),sort=TRUE,weights=TRUE)

    dBIC df weight
m2f  0.0 4  0.9412
m3f  5.5 6  0.0588
m0f 37.0 2  <0.001

> AICctab(m0f,m2f,m3f,delta=TRUE,nobs=nrow(orob1),sort=TRUE,weights=TRUE)

    dAICc df weight
m2f  0.0  4  0.99222
m3f  9.7  6  0.00778
m0f 35.8  2  < 0.001
```

Additions/enhancements/differences from `stats4::mle`

- `anova` method
- warnings on convergence failure
- more robust to non-positive-definite Hessian; can also specify `skip.hessian` to skip Hessian computation when it is problematic
- when profiling fails because better value is found, report new values
- can take named vectors as well as lists as starting parameter vectors
- added `AICc`, `BIC` definitions, `ICtab` functions
- added `"uniroot"` and `"quad"` options to `confint`
- more options for colors and line types etc etc. The old arguments are:

```
> function (x, levels, conf = c(99, 95, 90, 80, 50)/100, nseg = 50,
            absVal = TRUE, ...) {}
```

  The new one is:

```
> function (x, levels, which=1:p, conf = c(99, 95, 90, 80, 50)/100, nseg = 50,
            plot.confstr = FALSE, confstr = NULL, absVal = TRUE, add = FALSE,
            col.minval="green", lty.minval=2,
            col.conf="magenta", lty.conf=2,
            col.prof="blue", lty.prof=1,
            xlabs=nm, ylab="score",
            show.points=FALSE,
            main, xlim, ylim, ...) {}
```

  `which` selects (by character vector or numbers) which parameters to plot: `nseg` does nothing (even in the old version); `plot.confstr` turns on the labels for the confidence levels; `confstr` gives the labels; `add` specifies whether to add the profile to an existing plot; `col` and `lty` options specify the colors and line types for horizontal and vertical lines marking the minimum and confidence vals and the profile curve; `xlabs` gives a vector of x labels; `ylab` gives the y label; `show.points` specifies whether to show the raw points computed.

- `mle.options()`
- `data` argument
- handling of names in argument lists
- can use alternative optimizers (`nlminb`, `constrOptim`)

2 Newer stuff

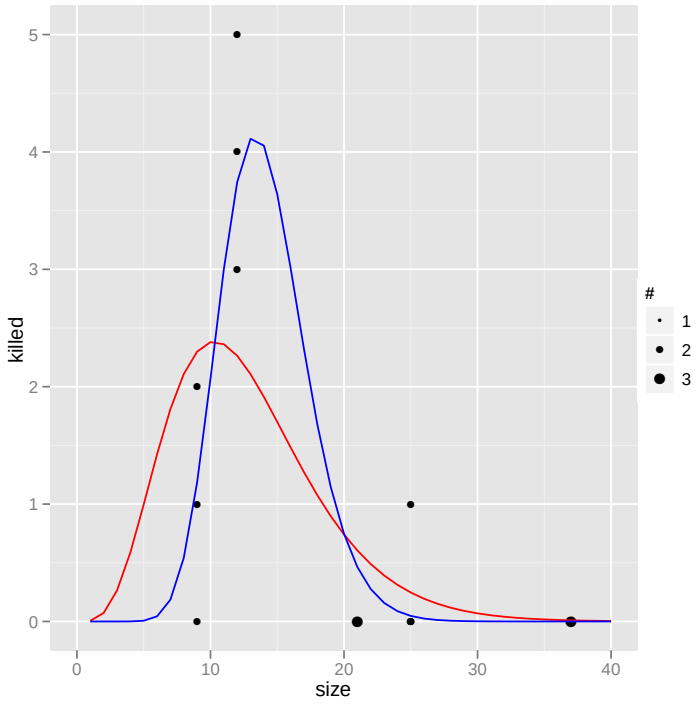
To do:

- use `predict`, `simulate` etc. to demonstrate different parametric bootstrap approaches to confidence and prediction intervals
  - use `predict` to get means and standard deviations, use delta method?
  - use `vcov`, assuming quadratic profiles, with `predict(...,newparams=...)`
  - prediction intervals assuming no parameter uncertainty with `simulate`
  - both together ...

3 Example

Data from an experiment by Vonesh (Vonesh and Bolker, 2005)

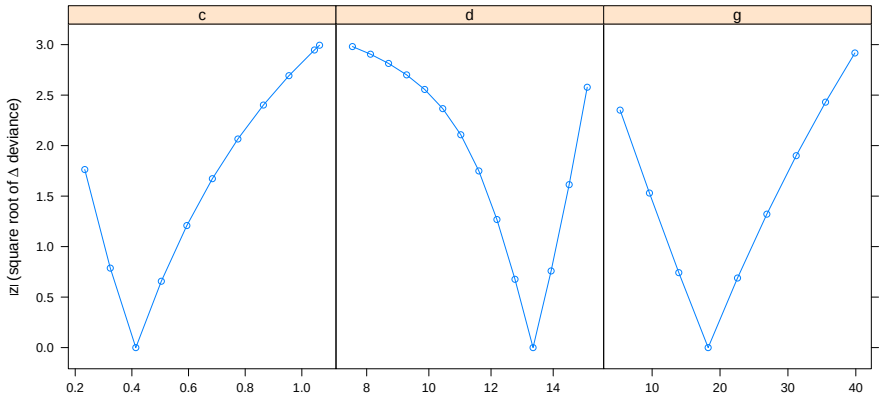
```
> frogdat <- data.frame(
   size=rep(c(9,12,21,25,37),each=3),
   killed=c(0,2,1,3,4,5,rep(0,4),1,rep(0,4)))
> frogdat$initial <- rep(10,nrow(frogdat))

> library(ggplot2)

> gg1 <- ggplot(frogdat,aes(x=size,y=killed))+geom_point()+
       stat_sum(aes(size=factor(..n..)))+
       labs(size="#")+scale_x_continuous(limits=c(0,40))

> m3 <- mle2(killed~dbinom(prob=c*(size/d)^g*exp(1-size/d),
   size=initial),data=frogdat,start=list(c=0.5,d=5,g=1))
> pdat <- data.frame(size=1:40,initial=rep(10,40))
> pdat1 <- data.frame(pdat,killed=predict(m3,newdata=pdat))

> m4 <- mle2(killed~dbinom(prob=c*((size/d)*exp(1-size/d))^g,
   size=initial),data=frogdat,start=list(c=0.5,d=5,g=1))
> pdat2 <- data.frame(pdat,killed=predict(m4,newdata=pdat))

> print(gg1 + geom_line(data=pdat1,colour="red")+geom_line(data=pdat2,colour="blue"))
```



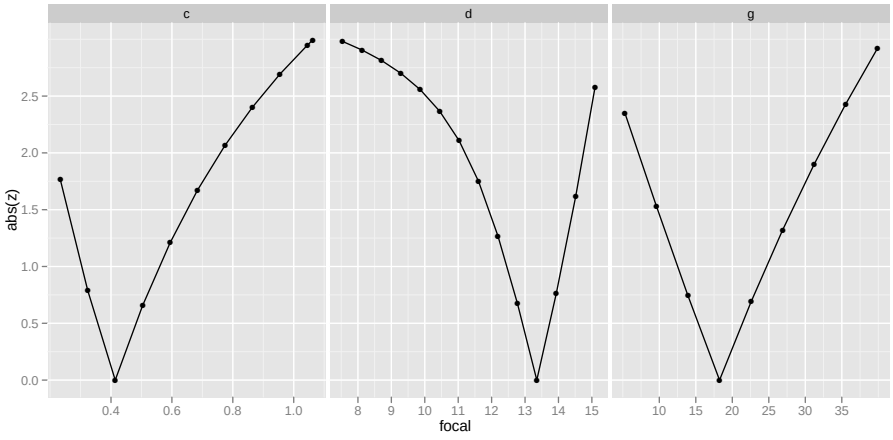

```
> coef(m4)

         c          d          g
 0.4138847 13.3517574 18.2511264

> prof4 <- profile(m4)
```

Three different ways to draw the profile:

```
> print(plot(prof4))

NULL

> prof4_df <- as.data.frame(prof4)
> library(lattice)
> print(xyplot(abs(z)~focal|param,data=prof4_df,
        subset=abs(z)<3,
        type="b",
        xlab="",
        ylab=expression(paste(abs(z),
            " (square root of ",Delta," deviance)")),
        scale=list(x=list(relation="free")),
              layout=c(3,1)))
```




```
> print(ggplot(subset(prof4_df,abs(z)<3),
       aes(x=focal,y=abs(z)))+geom_line()+
      geom_point()+
      facet_grid(~param,scale="free_x"))
```



Bugs, wishes, to do

- **WISH**: further methods and arguments: `subset`, `predict`, `resid`: `sim`?
- **WISH**: extend ICtab to allow DIC as well?
- minor **WISH**: better methods for extracting `nobs` information when possible (e.g. with formula interface)
- **WISH**: better documentation, especially for S4 methods

- **WISH**: variable-length chunks in argument list
- **WISH**: limited automatic differentiation (add capability for common distributions)

References

Crowder, M. J. (1978). Beta-binomial Anova for proportions. *Applied Statistics 27*, 34–37.

Vonesh, J. R. and B. M. Bolker (2005). Compensatory larval responses shift tradeoffs associated with predator-induced hatching plasticity. *Ecology 86*(6), 1580–1591.