Manual for explicit parallel peer code EPPEER

Bernhard A. Schmitt (Univ. Marburg, Germany)*
Rüdiger Weiner (Univ. Halle, Germany)†

August 24, 2012

EPPEER is a FORTRAN95 code with OpenMP parallelization solving non-stiff initial value problems of ODEs

$$y'(t) = f\big(t, y(t)\big),\ t \in [t_0, t_{end}], \quad y(t_0) = y_0 \in \mathbb{R}^n, \tag{1}$$

with explicit peer two-step methods of orders $3 \dots 9$ with parallelism across the method. It uses automatic stepsize control and provides continuous output of full order. The current version is intended for small-scale parallelism as on current desktop PCs with $2 \dots 8$ cores, no parallelization of the call to the right-hand side f is required. It is available at
`www.mathematik.uni-marburg.de/~schmitt/peer` and
`numerik.mathematik.uni-halle.de/forschung/software/`

1 Peer methods

Peer two-step methods solve the initial value problem through time steps from t_m to $t_{m+1} = t_m + h_m$, $m \geq 0$. In each step they employ s stages $Y_{m,i}$, $i = 1, \dots, s$ which have all the same accuracy and stability properties. The class of peer methods has been introduced by Schmitt and Weiner 2004 in [1]. The explicit methods used in EPPEER are given by the time step

$$Y_{m,i} = \sum_{j=1}^{s} b_{ij} Y_{m-1,j} + h_m \sum_{j=1}^{s} a_{ij} f(t_{m-1,j}, Y_{m-1,j}), \quad i = 1, \dots, s. \tag{2}$$

These steps are parallel since on the right-hand side only information from the previous time step is used, at $t_{m-1,j} = t_{m-1} + h_{m-1} c_j$, $j = 1, \dots, s$. The last step offset is $c_s = 1$ providing reference solutions at t_m, t_{m+1}. Still, the global error is $O(h_m^s)$ for all stages. The methods in EPPEER with $s = 4, 6, 8$ stages are from [3] and may use up to $s = 4, 6, 8$ cores parallel. The methods with $s = 3, 5, 7, 9$ stage are FSAL methods, see [4] and use $s - 1 = 2, 4, 6, 8$ parallel cores. However, due to some overhead in (2) one observes speed-ups near the

*schmitt@mathematik.uni-marburg.de
†ruediger.weiner@mathematik.uni-halle.de

number of cores only for rather expensive right-hand sides f. Dense output by interpolation has order $O(h^s)$ for all methods. However, only for the FSAL methods with odd stage numbers 3,5,7,9 it is also continuous due to the property $Y_{m-1,s} = Y_{m1}$.

The authors acknowledge helpful contributions from the coauthors Katja Biermann, Stefan Jebens, Helmut Podhaisky and from Matthias Korch.

2 Module structure

IVPEPP Contains the subroutine EPPEER and supporting subroutines in the file `ivpepp.f90`. All variables in this module are private, some may be accessed by supporting routines.

IVPRKP Using two-step peer methods the code needs a starting procedure for $Y_{0,i}$, $i = 1, \ldots, s$. Here the Runge-Kutta method DOPRI5 is used contained in the file `ivprkp.f90`.

ODEPROB It is convenient if the description of the initial value problem (1) is also contained in an own module with the problem size, start and end times and a subroutine fcn implementing the right-hand side f. More details in Section 3.2. Examples for this module are contained in the files `bruss2h.f90` with a 2D-reaction diffusion equation with the Brusselator (small diffusion constant), and the file `mbod4h.f90` with a multi-body problem for 400 masses. Only for the expensive problem `mbod4h.f90` speed-ups near the number of cores may be expected.

3 Calling sequences

The peer methods rely on the method coefficients a_{ij}, b_{ij} from (2), among others. In order to save the effort for their computation on multiple calls to EPPEER an initialization and deallocation of these data is required.

3.1 Minimal calling sequence

The *minimal calling sequence* is

```
   call ppsetcoeff(mnr,stages,mthn)  ! initializes method
!! define (repeatedly) initial values tm,te,ym and call:
   call eppeer(fcn,tm,te,ym,cpar)    ! call to integrator
   irep = ppreport(.true.)           ! prints error message
   call ppfreecoeff                  ! release memory
```

This sequence computes the numerical solution with automatic stepsize control with absolute and relative tolerances $atol = rtol = 1D-5$ and no dense output.

The declarations of the subroutines in use are

ppsetcoeff Allocates memory and computes method coefficients, declaration:

```
subroutine ppsetcoeff(mnr,pstage,methname)
  integer, intent(in) :: mnr
  integer, intent(out) :: pstage
  character(len=16), intent(out) :: methname
```

Parameters:

mnr	in	chooses method, available $mnr \in \{3,4,5,6,7,8,9\}$.
pstage	out	stage number s of the method, $pstage < 0$ indicates wrong value of mnr
methname	out	the name of the chosen method, it has the form **epp**s**xx**, where s is the number of stages. It may be used for naming output files as in `ivp_pmain.f90`.

eppeer performs time integration, declaration:

```
subroutine eppeer(fcn,t,tend,y,cpar,solout)
  interface
    pure subroutine fcn(t,u,udot,par)
     real(8), intent(in) :: t
     real(8), dimension(:), intent(in) :: u
     real(8), dimension(:), intent(out) :: udot
     real(8), dimension(:), intent(in) :: par
    end subroutine fcn
    subroutine solout(n,ts,tnew)
     integer, intent(in) :: n
     real(8) :: ts
     real(8), intent(in) :: tnew
    end subroutine solout
  end interface
  optional solout
  real(8) :: t,tend
  real(8), dimension(:) :: y
  real(8), dimension(:), intent(in) :: cpar
```

Parameters:

fcn	sub	this is a subroutine implementing the right-hand side f from (1), where t is the time, u is the input vector, and $udot = f(t, u)$. It is declared as a pure Fortran95 subroutine with no side effects and no internal parallelism. The vector *par* may contain additional parameters passed by the argument *cpar* from the calling program through EPPEER.
t	in/out	contains start time t_0 from (1) on call, and end time on return
tend	in	end time t_{end} of integration
y	in/out	contains initial value y_0 from (1) on call, and numerical solution $y(t_{end})$ on return. The length $n = size(y)$ of this vector is considered to be the dimension of the ODE problem
cpar	in	dummy variable, vector of parameters passed to fcn in every call
solout	sub	optional, subroutine for dense output, see description of subroutine *ppcont* in the next subsection.

ppreport function returns exit status of call to EPPEER, computation successful for PPREPORT()=0.

```
function ppreport(prterr)
logical, intent(in), optional :: prterr
integer :: ppreport
```

If the optional argument is given with *prterr* = *.true.*, error messages are printed to the console. The error codes are:

-1	error: missing initialization with *ppsetparam*
1	failure, step size too small, $hs < hmin = 1D - 10$
2	failure, too many steps, $nsteps > maxsteps = 10^6$.

ppfreecoeff deallocates all matrices declared by PPSETCOEFF.

The additional parameters *cpar* and *par* are presently not used by EPPEER but are introduced for convenience and for upward compatibility to a later version computing also solution sensitivities, see a forthcoming paper of Schmitt and Kostina in SINUM.

3.2 Information supplied by user

On calling EPPEER the user has to supply all information on the initial value problem (1), i.e. start and end times and initial value y_0 in y with $size(y) = n$, as explained in Section 3.1. The subroutine FCN for the function $f(t, u)$ has to be declared with the header

```
pure subroutine fcn(t,u,udot,par)
 real(8), intent(in) :: t
 real(8), dimension(:), intent(in) :: u
 real(8), dimension(:), intent(out) :: udot
 real(8), dimension(:), intent(in) :: par
```

The attribute "pure" simplifies parallelization for the compiler and means that the subroutine has no side effects like common blocks. For convenience the example files `bruss2h.f90`, `mbod4h.f90` from the package both define such a module ODEPROB and provide additional information used by the driver main program `ivp_pmain.f90`. These informations (not directly used by EPPEER) are

nprob	integer	problem dimension, it is passed to EPPEER through $size(y)$.
nparm	integer	dimension of parameter array *par* in fcn
odename	char	name of the example problem (used for naming output files)
exsol	logical	.true. if exact solution at t_{end} is known
inivals	sub	CALL INIVALS$(t0, te, u, par)$ provides the data $t_0 = t0$, $t_{end} = te$ and $y_0 = u$
solution	sub	CALL SOLUTION(t, u, par) provides the exact solution $y(t)$ if $exsol = .true.$.

3.3 Control inputs and additional subroutines

The operation of EPPEER may be influenced by additional subroutines contained in the module `ivpepp.f90`. Some provide also additional information about its performance and the solution. The subroutines PPSETACC and PPSETJOB must be called after the initialization by PPSETCOEFF and before EPPEER, the subroutine PPGETSTATS after EPPEER.

ppsetjob Sets control switches

```
subroutine ppsetjob(stepcon, contout)
logical, intent(in), optional :: stepcon, contout
```

stepcon	in	switches stepsize control on/off, default is on, $stepcon = .true.$
contout	in	switches dense output on/off, default is off, $contout = .false.$. Dense output calls the optional subroutine SOLOUT. If it is missing in the call to EPPEER no output is called. More details are given below with the subroutine PPCONT.

ppsetacc Set accuracy of numerical solution

```
subroutine ppsetacc(atol,rtol,hstep)
real(8), intent(in) :: atol, rtol
real(8), intent(in), optional :: hstep
```

The parameters are

atol	in	sets absolute tolerance for error control
rtol	in	sets relative tolerance; the error estimate err_i of the numerical solution will satisfy $\lvert err_i\rvert/(atol+rtol\lvert y_i\rvert) \leq 1$ in every component i if $stepcon = .true.$
hstep	in	optional, initial stepsize if $stepcon = .true.$, or fixed stepsize for integration without stepsize control, $stepcon = .false.$. For $hstep \leq 0$ step control is used, anyway.

ppgetstats returns integrator statistics

```
subroutine ppgetstats(nsteps,nrej,nfcn)
integer, intent(out) :: nsteps, nrej, nfcn
```

where

nsteps	out	total number of time steps (including rejected steps)
nrej	out	number of rejected steps (contained in *nsteps*)
nfcn	out	total number of calls to FCN (including Runge-Kutta startup)

ppcont Computes dense output by Lagrangian interpolation of the most recent stages $Y_{m,i}$, $i = 1, \ldots, s$.

```
subroutine ppcont(tsol,tnew,ycon)
real(8), intent(in) :: tsol,tnew
real(8), dimension(:), intent(out) :: ycon
```

Parameters:

tsol	in	point where solution is to be evaluated: $y(tsol)$. The point *tsol* should lie in the interval $[t_m, t_{m+1}]$ of the current time step
tnew	in	end point t_{m+1} of current time step, the start point t_m is known to *ppcont* through a private variable
ycon	out	receives the solution approximation for $y(tsol)$

Usage: PPCONT may be called by the user-supplied subroutine SOLOUT from the parameter list of EPPEER. EPPEER initializes a variable $tsol = t_0$ and, if $pcout = .true.$, calls SOLOUT after every successful time step with the parameter list $\text{SOLOUT}(size(y), tsol, t + hs)$, where $t + hs = t_{m+1}$.

On each call SOLOUT may repeatedly compute solution values by calling PPCONT($tsol, tnew, ycon$) and increasing $tsol$ until $tsol > tnew$. An example of such a subroutine is given by

```
subroutine psolout(n,tsol,tnew)
  use ivpepp
  implicit none
  integer, intent(in) :: n
  real(8) :: tsol
  real(8), intent(in) :: tnew
  real(8), dimension(n) :: ycon
  do while (tsol<=tnew)
    call ppcont(tsol,tnew,ycon)
    write(4,"(F12.3,2E14.5)") tsol,ycon(1),ycon(2)
    tsol = tsol+0.05D0
  end do
end subroutine psolout
```

This subroutine is contained in `ivp_pmain.f90`, it computes the solution at equidistant points $0.05i$, $i \geq 0$, and writes its first two components to a text file. The while loop insures that the calls to *ppcont* stop if $tsol > tnew = t_{m+1}$.

4 GFortran compiler and OpenMP

Parallelization is obtained with OpenMP (see *www.openmp.org*) by invoking the `-fopenmp` switch in Fortran compilers supporting OpenMP. The package has been tested with the free GNU gfortran compiler (see *gcc.gnu.org/fortran/*) under Windows 7. A simple commando sequence to start the driver program `ivp_pmain.f90` with the multi-body example `mbod4h.f90` is

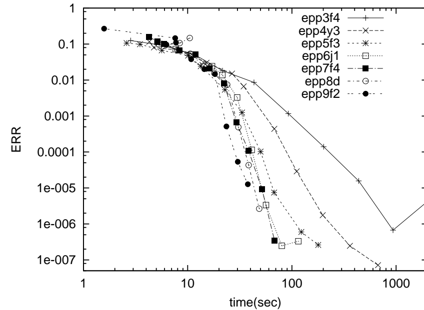
```
gfortran -c mbod4h.f90
gfortran -c ivprkp.f90
gfortran -c -fopenmp ivpepp.f90
gfortran -fopenmp ivprkp.o ivpepp.o mbod4h.o ivp_pmain.f90
a
```

The first command compiles the specific example of the module ODEPROB and generates the Fortran module file `odeprob.mod` and the object file `mbod4h.o`. Note, that only the integrator module `ivpepp.f90` with the EPPEER subroutine and the main program `ivp_pmain.f90` use OpenMP parallelization.

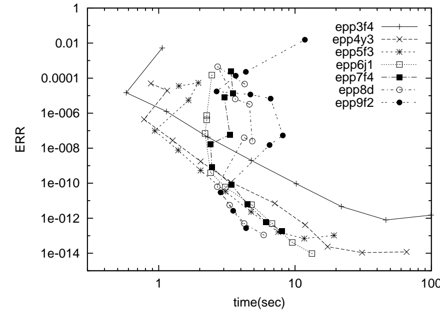
4.1 Performance

All seven peer methods provided by EPPEER have been tested on a PC with Intel i7-860 Quadcore (with hyperthreading) with 2.8 GHz. The driver program

ivp_pmain.f90 computes solutions for one chosen peer method with tolerances $atol = rtol = 10^{-2}, 10^{-3}, \ldots, 10^{-12}$ and writes accuracies and computing times to a log file with a name combined from the number of threads, *odename* (§3.2) and *methname* (§3.1). The following Gnuplot diagrams depict the efficiency of these methods by showing the computing times (OpenMP thread times) in relation to the achieved accuracies (exact errors) at t_{end}.



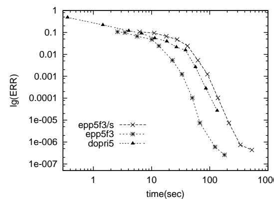
all peer methods on MBOD4h



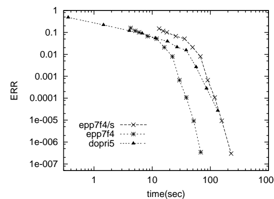
all peer methods on BRUSS2h

Notes: MBOD4h is a difficult problem with expensive calls to fcn, all methods miss the tolerances quite far. Still, the efficiency obviously improves with increasing stage numbers and orders of the methods.
BRUS2h is a mildly stiff problem and only for sharp tolerances higher order pays off.

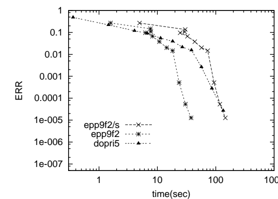
Parallel performance of EPPEER is best for problems with expensive function call fcn as it is the case for example MBOD4h. For this problem the next diagrams compare the parallel run times of individual methods with the sequential runtime of the same method (no `-fopenmp` switch) and the sequential runtime of the Runge-Kutta code DOPRI5 of Dormand/Prince.



epp5f3 and dopri5



epp7f3 and dopri5



epp9f2 and dopri5

Speed-up for epp5f3 is up to 3.1, for epp7f3 it is between 3 and 3.3 and for epp9f2 it reaches 4.

5 Files in the EPPEER package

The archive eppeer.zip contains the following files.

`ivpepp.f90`	•	Module IVPEPP with integrator EPPEER
`ivprkp.f90`	•	Module IVPRKP, used by EPPEER
`mbod4h.f90`	○	Multi-body problem example for module ODEPROB
`bruss2h.f90`	○	Brusselator example for module ODEPROB
`ivp_pmain.f90`	○	Main program computing the numerical solution for different tolerances by calls to EPPEER, writes log files with errors and computing times. Log files may be used to produce efficiency diagrams with Gnuplot.
`man_epp.pdf`		This documentation
`mbod.plt`		Gnuplot command file producing an efficiency diagram for the multi-body example
`brus.plt`		Gnuplot command file producing an efficiency diagram for the Brusselator example

Bullets indicate that the file is required in the form provided, circles mean that the user should replace the file with an own one.

References

[1] B.A. Schmitt, and R. Weiner, *Parallel two-step W-methods with peer variables*, SIAM J. Numer. Anal. 42 (2004), 265-282.

[2] R. Weiner, K. Biermann, B.A. Schmitt, and H. Podhaisky, *Explicit two-step peer methods*, Comput. Math. Appl., 55 (2008), 609-619.

[3] B.A. Schmitt, R. Weiner, and S. Jebens, *Parameter optimization for explicit parallel peer two-step methods*, Appl. Numer. Math., 59 (2009), 769-782.

[4] B.A. Schmitt, R. Weiner, and St. Beck, *Two-step peer methods with continuous output*, Bericht 2012-02, FB Mathematik u. Informatik, Univ. Marburg, see
www.uni-marburg.de/fb12/forschung/berichte/berichtemathe/pdfbfm/bfm12-02.pdf