

## Re: help about ARPACK solver

---

*Source:* <http://sci.tech-archive.net/Archive/sci.math.num-analysis/2007-05/msg00168.html>

---

- *From:* andy2O <[andy2O@xxxxxxxxxxxx](mailto:andy2O@xxxxxxxxxxxx)>
  - *Date:* 21 May 2007 13:51:10 -0700
- 

On 21 May, 18:00, lorin <[wwang...@xxxxxxxxxx](mailto:wwang...@xxxxxxxxxx)> wrote:

On May 20, 7:31 am, andy2O <[and...@xxxxxxxxxxxx](mailto:and...@xxxxxxxxxxxx)> wrote:

On 16 May, 10:54, lorin  
<[wwang...@xxxxxxxxxx](mailto:wwang...@xxxxxxxxxx)>  
wrote:

[snip]

Hello,

I used the "parallel" vector to verify if the result satisfies  $A * x = \lambda * x$ . For the eigenvector from Matlab,  $(A * x - \lambda * x)$  could be as precise as (E-14), however, when I used the eigenvector from my fortran program (yes, there is a constant factor between the two eigenvectors),  $(A * x - \lambda * x)$  can only be (E-04), sometimes even (E-03), I think such precision can not be satisfied in our application.

Thanks all the same,

With my best wishes,

Re: help about ARPACK solver

lorin

Hi lorin,

[snip]

2) Matlab's eigs() function uses ARPACK, so logically your Fortran code *should* be able to solve the problem as accurately as Matlab *if* you get everything set up right... I don't know ARPACK well enough to give detailed help to you, but:

– Matlab uses 'double precision' variables throughout by default. Fortran uses single precision variables by default. If you want accurate results from your Fortran code you *must* use double precision. So:

(a) make sure you declare all your Fortran variables to be double precision – note that if you rely of Fortran's implicit typing mechanism this won't happen.

(b) remember that if you type a number such as 1.234 or 5.67e8 in a Fortran program, that is *single* precision. If you want double precision you need to use 1.234D0 or 5.67D8. This is a common cause of loss of precision in Fortran, particularly in test case code.

Re: help about ARPACK solver

If you identify any detailed Fortran problems, try asking in comp.lang.fortran – they're very helpful.

[snip]

andy

I've just remembered – you're using complex arithmetic.

Note that a 'COMPLEX' variable in Fortran is by default single precision (In fact in FORTRAN77, there is no double precision complex variable, but many compilers do support it as an extension). Fortran90 and later support double precision complex as standard, but it's still easy to accidentally use single precision COMPLEX variables. For example COMPLX(1.0D0,2.0D0) will return a single precision complex number!! I certainly find it confusing....

If you're unsure about the precision of your complex Fortran variables, read below (taken from [http://gcc.gnu.org/onlinedocs/gcc-3.4.6/g77/CMPLX\\_0028\\_0029-of-DOUBLE...](http://gcc.gnu.org/onlinedocs/gcc-3.4.6/g77/CMPLX_0028_0029-of-DOUBLE...)) and do some more research.

Yours,  
andy

In accordance with Fortran 90 and at least some (perhaps all) other compilers, the GNU Fortran language defines CMPLX() as always returning a result that is type COMPLEX(KIND=1).

This means `CMPLX(D1,D2)', where `D1' and `D2' are REAL(KIND=2) (DOUBLE PRECISION), is treated as:

CMPLX(SNGL(D1), SNGL(D2))

Re: help about ARPACK solver

(It was necessary for Fortran 90 to specify this behavior for DOUBLE PRECISION arguments, since that is the behavior mandated by FORTRAN 77.)

The GNU Fortran language also provides the DCMPLX() intrinsic, which is provided by some FORTRAN 77 compilers to construct a DOUBLE COMPLEX entity from of DOUBLE PRECISION operands. However, this solution does not scale well when more COMPLEX types (having various precisions and ranges) are offered by Fortran implementations.

Fortran 90 extends the CMPLX() intrinsic by adding an extra argument used to specify the desired kind of complex result.

I just verify the eigenvectors from my Fortran program, the results is exciting.

As Andy said, I think the whole problem's cause lies in the precision loss. I write

a small program to compute  $A*x - \lambda*x$  under Fortran environment, the result is OK

(E-14 ~ E-15), thanks, Andy, your advice is really helpful.

However, I still dont know what kind of criteria Matlab adopts to form the eigenvector,

why exists such a mismatch between my Fortran program and Matlab, though they both use Arpack solver.

With my best wishes to all,

lorin

I'm pleased to hear your code works!

Personally, I cannot see too much reason to worry about the 'mismatch' between the Fortran and Matlab results, as long as you can do tests and the results are reliably correct:

– As long as the vectors are parallel then both sets of eigenvectors can be equally correct. I don't think you can say one is 'better' than the other in any sense!! Do you have any reason to prefer one to the other?

–Perhaps it is to do with the initial guess used to start the iteration? I don't have access to Matlab easily, so I cannot test this myself. But perhaps you can run the Fortran code from two initial

Re: help about ARPACK solver

guesses and see if the two Fortran results give 'different' parallel eigenvectors. Then try the same with Matlab with different v0 options.

– Perhaps Matlab uses a slightly older version of ARPACK, or a slightly newer one? Or perhaps Matlab make a few changes or improvements to the standard version of ARPACK before they put it into their own code. Unfortunately this is hard to find out, because Matlab is a commercial code.

I'd say you should keep testing, but if the test results are OK then don't worry too much about the differences between the two codes.

Bye for now, and good luck with the rest of your work!

Yours,  
andy

.