Subject: Challenging question - array curve fitting
Posted by Qing on Tue, 27 Mar 2007 08:37:38 GMT
View Forum Message <> Reply to Message

G'day folks,

I have a time series of images represented by a 3D array as Data(Nx, Ny, Nt) or Data (Nt, Nx, Ny). I would like to apply a non-linear curve fitting to the time dimension for every pixel respectively. I can loop through every pixel using 1-D curve fitting procedure, but the process is slow and it does not make efficient use of multiple CPUs.

Theoretically I would think it should be feasible to perform curve fitting for all pixels simultaneously via matrix operation? However, all the IDL's fitting routines only accept vectors for input parameters to my knowledge. Does anyone know if there is any non-linear fitting routines that accept array parameters. Or can anyone comment on whether such a routine is feasible at all?

Qing ;-?

Subject: Re: Challenging question - array curve fitting
Posted by Craig Markwardt on Thu, 29 Mar 2007 15:07:38 GMT
View Forum Message <> Reply to Message

"Qing" <csis@bigpond.net.au> writes:

> G'day folks,
>
> I have a time series of images represented by a 3D array as Data(Nx,
> Ny, Nt) or Data (Nt, Nx, Ny). I would like to apply a non-linear curve
> fitting to the time dimension for every pixel respectively. I can loop
> through every pixel using 1-D curve fitting procedure, but the process
> is slow and it does not make efficient use of multiple CPUs.
>
> Theoretically I would think it should be feasible to perform curve
> fitting for all pixels simultaneously via matrix operation? However,
> all the IDL's fitting routines only accept vectors for input
> parameters to my knowledge. Does anyone know if there is any non-
> linear fitting routines that accept array parameters. Or can anyone
> comment on whether such a routine is feasible at all?

Greetings, there should be nothing stopping you from grouping multiple time series into a single large vector, and fitting them simultaneously. You just need to make your model function smart enough to know what to do with the concatenated data set.

However, there is a point of diminishing returns. Since the number of arithmetic operations required to perform the fit scales as the number of pixels *cubed*, there is really no advantage to grouping large numbers of pixels together, in fact there may be a disadvantage. This depends on the number of times (your Nt), but since we don't know that number, you will have to find the right balance yourself.

Good luck,
Craig

--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@REMOVEcow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: Challenging question - array curve fitting
Posted by Qing on Tue, 03 Apr 2007 13:31:44 GMT
View Forum Message <> Reply to Message

On Mar 30, 1:07 am, Craig Markwardt
<craigm...@REMOVEcow.physics.wisc.edu> wrote:
> "Qing" <c...@bigpond.net.au> writes:
>> G'day folks,
>
>> I have a time series of images represented by a 3D array as Data(Nx,
>> Ny, Nt) or Data (Nt, Nx, Ny). I would like to apply a non-linear curve
>> fitting to the time dimension for every pixel respectively. I can loop
>> through every pixel using 1-D curve fitting procedure, but the process
>> is slow and it does not make efficient use of multiple CPUs.
>
>> Theoretically I would think it should be feasible to perform curve
>> fitting for all pixels simultaneously via matrix operation? However,
>> all the IDL's fitting routines only accept vectors for input
>> parameters to my knowledge. Does anyone know if there is any non-
>> linear fitting routines that accept array parameters. Or can anyone
>> comment on whether such a routine is feasible at all?
>
> Greetings, there should be nothing stopping you from grouping multiple
> time series into a single large vector, and fitting them
> simultaneously. You just need to make your model function smart
> enough to know what to do with the concatenated data set.
>
> However, there is a point of diminishing returns. Since the number of
> arithmetic operations required to perform the fit scales as the number
> of pixels *cubed*, there is really no advantage to grouping large

> numbers of pixels together, in fact there may be a disadvantage. This
> depends on the number of times (your Nt), but since we don't know that
> number, you will have to find the right balance yourself.
>
> Good luck,
> Craig
>
> --
> ----------
> Craig B. Markwardt, Ph.D. EMAIL: craigm...@REMOVEcow.physics.wisc.edu
> Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
> ----------

Hello Craig,

Thanks a lot for your comments and tips. It is intriguing for
"grouping multiple
time series into a single large vector...". I can manage to transform/
reform the
data array into a large vector, but my brain just can't think of a way
to model
the concatenated vector independently. For example, I am using a
Gaussian curve
model with 3 fitting parameters for each curve. Typically Nx=Ny=128
and the
number of time points Nt=60. The thing is that my computer has two
CPUs, and it
only uses about 50% total CPU when fitting the curve by looping
through each pixel.
I though usually array operation is more efficient than looping throug
all elements individually, but I was not sure if that is the case for
a non-linear
fitting task. Or at least, using array operation can get better use of
the CPUs
upto 100%. Do you thing using a large vector would be as efficient as
using array?
Why does "the number of arithmetic operations required to perform the
fit scales
as the number of pixels *cubed*"?, I thought it would be a linear
relation if using
array just like looping through all pixels one-by-one. Am I missing
something?

I would really appreciate any further elaboration.

Puzzled from Downunder
Qing

Subject: Re: Challenging question - array curve fitting
Posted by JD Smith on Tue, 03 Apr 2007 20:25:10 GMT
View Forum Message <> Reply to Message

Just an unrelated followup question for Craig... MPFIT is (I hate to say it) quite slow, in particular for problems like this, where you want to simultaneously fit hundreds or thousands of things to complex models. I know it wasn't designed for speed, and the things it does are amazing (I particularly like arbitrary constraint expressions), but I wonder, what do you think the chances of improving the speed are? Would it require a fundamental rewrite, or is it more sensible to fall back on a C or FORTRAN version of the library (I actually don't know how the speed compares)? Should we lobby ITTVIS to throw out their non-linear fitters, and incorporate MPFIT into compiled code? Just want to get your perspectives. Thanks again for such a wonderfully useful tool.

JD

Subject: Re: Challenging question - array curve fitting
Posted by Craig Markwardt on Mon, 23 Apr 2007 02:49:56 GMT
View Forum Message <> Reply to Message

Sorry for neglecting your post for so long!

"Qing" <csis@bigpond.net.au> writes:
> Hello Craig,
>
> Thanks a lot for your comments and tips. It is intriguing for
> "grouping multiple time series into a single large vector...". I can
> manage to transform/ reform the data array into a large vector, but
> my brain just can't think of a way to model the concatenated vector
> independently. For example, I am using a Gaussian curve model with 3
> fitting parameters for each curve. Typically Nx=Ny=128 and the
> number of time points Nt=60. The thing is that my computer has two
> CPUs, and it only uses about 50% total CPU when fitting the curve by
> looping through each pixel.

Your model function would still need to compute each light curve separately, which may involve a loop. But, for example, you could loop over time sample instead of light curve number, and in each iteration compute 128x128 model values at once (or fewer).

Example:
; Compute NX x NY x NT light curve samples
; Model is simple linear P0 + P1*T
; Parameters are arranged like this:

```
;    P0 = P(0:(NX*NY-1))  ;; For each pixel
;    P1 = P(NX*NY:*)      ;; For each pixel
function lcmod, t, p, nx=nx, ny=ny
  ntot = nx*ny
  p0 = reform(p(0:ntot-1),nx,ny)
  p1 = reform(p(ntot-1:*),nx,ny)
  nt = n_elements(t)
  model = fltarr(nx,ny,nt)
  for i = 0, nt-1 do model(0,0,i) = p0 + p1*t(i)
  return, model
end
```

This only works because NX*NY is much larger than NT.

> I though usually array operation is more efficient than looping
> throug all elements individually, but I was not sure if that is the
> case for a non-linear fitting task. Or at least, using array
> operation can get better use of the CPUs upto 100%. Do you thing
> using a large vector would be as efficient as using array?

It all depends on how much work is done per iteration of the loop. If you can accomplish a lot of work in one iteration, then you will not save by vectorizing the loop. Since MPFIT has a lot of set-up and tear-down expenses, then I suspect you could indeed gain by grouping a several time series together.

> Why does "the number of arithmetic operations required to perform
> the fit scales as the number of pixels *cubed*"?, I thought it would
> be a linear relation if using array just like looping through all
> pixels one-by-one. Am I missing something?

Actually it scales as M N^2 where M is the number of data points and N is the number of parameters. However, since this example involves grouping independent light curves with independent parameters into one block, M is also proportional to N, hence an overall N^3 dependence.

Hope you succeeded!
Craig

--

---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@REMOVEcow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: Challenging question - array curve fitting
Posted by Craig Markwardt on Mon, 23 Apr 2007 03:50:39 GMT
View Forum Message <> Reply to Message

Here's a late follow-up to your questions.

JD Smith <jdsmith@as.arizona.edu> writes:
> Just an unrelated followup question for Craig... MPFIT is (I hate to
> say it) quite slow, in particular for problems like this, where you
> want to simultaneously fit hundreds or thousands of things to complex
> models. I know it wasn't designed for speed, and the things it does
> are amazing (I particularly like arbitrary constraint expressions),
> but I wonder, what do you think the chances of improving the speed
> are? Would it require a fundamental rewrite, or is it more sensible
> to fall back on a C or FORTRAN version of the library (I actually
> don't know how the speed compares)? Should we lobby ITTVIS to throw
> out their non-linear fitters, and incorporate MPFIT into compiled
> code? Just want to get your perspectives. Thanks again for such a
> wonderfully useful tool.

JD, at some point early on, I tried to squeeze as much performance out of MPFIT as I could. I imagine that today's processors are not the same as then, but the difference is not likely to be huge.

I did a simple test of fitting a random vector with a polynomial,

```
yy = randomn(seed,1024L*1024L)
profiler
p = mpfitfun('quad', dindgen(1024L*1024L), yy, 1d, [1d,1d,1d], perror=dp)
profiler, /report
```

and the results were 30% of the time was spent evaluating the polynomial function, 38% spent factoring the jacobian, 15% calculating vector norms. Even if we could make MPFIT run infinitely fast, it could only ever be three times faster. The factorization is the biggest proportion of time spent, but then it needs to be.

I think there are several factors to consider.

* If your function is very complicated and time consuming, then it won't matter how fast MPFIT is. My example shows that even a very simple quadratic model function with few parameters, already takes 30% of the total execution time to evaluate.

* The function evaluation speed can be significantly improved if you can compute analytical derivatives.

* If your fit takes too many iterations to converge, then you can relax the convergence criteria (ftol, xtol, gtol).

* If you're someone like the original poster that has many fits to perform, with a small number of points per fit, then it might indeed pay off to group multiple fits together, so you don't incur the setup and teardown costs of each MPFIT call.

In the meantime I have developed a C library version of the fitter. I have never tested whether it was faster, since that wasn't the point (C was a delivery requirement to a larger project).

Happy fitting!
Craig

P.S.
```
function quad, x, p
  return, p[0] + p[1]*x + p[2]*x*x
end
```

--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@REMOVEcow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: Challenging question - array curve fitting
Posted by Qing on Tue, 24 Apr 2007 12:49:25 GMT
View Forum Message <> Reply to Message

On Apr 23, 12:49 pm, Craig Markwardt
<craigm...@REMOVEcow.physics.wisc.edu> wrote:
> Sorry for neglecting your post for so long!
>
> "Qing" <c...@bigpond.net.au> writes:
>> Hello Craig,
>
>> Thanks a lot for your comments and tips. It is intriguing for
>> "grouping multiple time series into a single large vector...". I can
>> manage to transform/ reform the data array into a large vector, but
>> my brain just can't think of a way to model the concatenated vector
>> independently. For example, I am using a Gaussian curve model with 3
>> fitting parameters for each curve. Typically Nx=Ny=128 and the
>> number of time points Nt=60. The thing is that my computer has two
>> CPUs, and it only uses about 50% total CPU when fitting the curve by
>> looping through each pixel.
>
> Your model function would still need to compute each light curve
> separately, which may involve a loop. But, for example, you could

> loop over time sample instead of light curve number, and in each
> iteration compute 128x128 model values at once (or fewer).
>
> Example:
> ; Compute NX x NY x NT light curve samples
> ; Model is simple linear P0 + P1*T
> ; Parameters are arranged like this:
> ; P0 = P(0:(NX*NY-1)) ;; For each pixel
> ; P1 = P(NX*NY:*) ;; For each pixel
> function lcmod, t, p, nx=nx, ny=ny
> ntot = nx*ny
> p0 = reform(p(0:ntot-1),nx,ny)
> p1 = reform(p(ntot-1:*),nx,ny)
> nt = n_elements(t)
> model = fltarr(nx,ny,nt)
> for i = 0, nt-1 do model(0,0,i) = p0 + p1*t(i)
> return, model
> end
>
> This only works because NX*NY is much larger than NT.
>
>> I though usually array operation is more efficient than looping
>> throug all elements individually, but I was not sure if that is the
>> case for a non-linear fitting task. Or at least, using array
>> operation can get better use of the CPUs upto 100%. Do you thing
>> using a large vector would be as efficient as using array?
>
> It all depends on how much work is done per iteration of the loop. If
> you can accomplish a lot of work in one iteration, then you will not
> save by vectorizing the loop. Since MPFIT has a lot of set-up and
> tear-down expenses, then I suspect you could indeed gain by grouping a
> several time series together.
>
>> Why does "the number of arithmetic operations required to perform
>> the fit scales as the number of pixels *cubed*"?, I thought it would
>> be a linear relation if using array just like looping through all
>> pixels one-by-one. Am I missing something?
>
> Actually it scales as M N^2 where M is the number of data points and N
> is the number of parameters. However, since this example involves
> grouping independent light curves with independent parameters into one
> block, M is also proportional to N, hence an overall N^3 dependence.
>
> Hope you succeeded!
> Craig

Hi Craig,

Champion! Thanks you soooooo much for the tips. I will try it to see
if this
can speed up my curve fittings!

Cheers :-))