Subject: How to speed up KRIG2D by 30x
Posted by Mike[5] on Tue, 08 Oct 2013 14:51:57 GMT
View Forum Message <> Reply to Message

While trying to figure out what the KRIG2D routine was doing, I discovered it can be speed up by one order of magnitude or more, for large numbers of output points.

One only needs to change a single line of code.

I post this here hoping someone at EXELIS will see it and include in the next IDL distribution...

In practice the last few lines of the current KRIG2D routine (as of IDL 8.2.3) should be changed from this

```
;+++++++++++++++++++++++++++++++
for j=0,ny-1 do begin              ;Each output point
  y0 = bounds[1] + gs[1] * j
  for i=0,nx-1 do begin
    x0 = bounds[0] + gs[0] * i
    d[0] = reform(sqrt((x-x0)^2 + (y-y0)^2),n) ;distance
    d = call_function(fname, d, t)          ;Get rhs
    d[n] = 1.0                           ;lagrange constr
    d=LUSOL( a, indx, d)                  ;Solve using LUDC results.
    r[i,j] = total(d * z)
  endfor
endfor
;-------------------------------
```

into the following mathematically equivalent form, with LUSOL outside the double FOR loop and a change in the order of the array operations. This works because array multiplication is commutative.

```
;+++++++++++++++++++++++++++++++
az = LUSOL(a, indx, [z,0.0])               ;Solve using LUDC results.
for j=0,ny-1 do begin                      ;Each output point
  y0 = bounds[1] + gs[1] * j
  for i=0,nx-1 do begin
    x0 = bounds[0] + gs[0] * i
    d[0] = reform(sqrt((x-x0)^2 + (y-y0)^2),n) ;distance
    d = call_function(fname, d, t)          ;Get rhs
    d[n] = 1.0                           ;lagrange constr
    r[i,j] = total(d * az)
  endfor
endfor
;-------------------------------
```

In the following example, with a large number of output points, my modified routine KRIG2D_MODIFIED is 30x faster than IDL's KRIG2D and also 7x faster than the native code

GRIDDATA. This suggests that GRIDDATA is making the same mistake as KRIG2D of including the back substitution inside the final for loop.

```
;++++++++++++++++++++++++++++++
; Create a dataset of N points.
n = 500 ;# of scattered points
seed = -121147L ;For consistency
x = RANDOMU(seed, n)
y = RANDOMU(seed, n)

; Create a dependent variable in the form a function of (x,y)
z = 3 * EXP(-((9*x-2)^2 + (7-9*y)^2)/4) + $
   3 * EXP(-((9*x+1)^2)/49 - (1-0.9*y)) + $
   2 * EXP(-((9*x-7)^2 + (6-9*y)^2)/4) - $
   EXP(-(9*x-4)^2 - (2-9*y)^2)

s = [0.5,0]

tic
z2 = krig2d_modified(z, x, y, EXPONENTIAL=s, NX=200, NY=200)
toc

tic
z2 = GRIDDATA(x, y, z, /KRIGING, DIMENSION=[200,200], VARIOGRAM=[2,s])
toc

tic
z2 = krig2d(z, x, y, EXPONENTIAL=s, NX=200, NY=200)
toc
;------------------------------
```

Subject: Re: How to speed up KRIG2D by 30x
Posted by timothyja123 on Wed, 09 Oct 2013 02:59:54 GMT
View Forum Message <> Reply to Message

You should probably put in a support request to Exelis rather than just hoping they see this post. That way it will be in their system and is less likely to get lost and forgotten about.

Subject: Re: How to speed up KRIG2D by 30x
Posted by chris_torrence@NOSPAM on Wed, 09 Oct 2013 17:32:53 GMT
View Forum Message <> Reply to Message

Hi Mike,

This is fantastic. I'm working on adding in your change. However, I just found a different problem.

If you look at the "Krig_Sphere" function within krig2d.pro, the code doesn't match the docs. The documentation states that for spherical covariance:

```
C(d) = C1 - 1.5 C1 (d/A) + 0.5 C1 (d/A)^3  if d < A
     = C0 + C1    if d = 0
     = 0          if d > A
```

Note that I threw in a factor of C1 on the first line (the docs are wrong).

Here is the code:

```
FUNCTION Krig_sphere, d, t
  r = d/t[0]
  v = t[1] + t[2] * (r * (1.5 - 0.5 * r *r) > 0)
  z = where(d eq 0, count)
  if count ne 0 then v[z] = 0
  return, (t[1] + t[2]) - v
end
```

We are not clipping to 0 for d > A. In fact, for d > A, the function actually starts to go back up and levels off at the constant C1. You can see this with the following plot:

```
p = plot(krig_sphere(findgen(40), [10, 0.5, 1]))
```

I think the code should be something more like:

```
FUNCTION Krig_sphere, d, t
  r = d/t[0]
  v = t[2] * (1 - r * (1.5 - 0.5*r*r))
  v[WHERE(d eq 0, /NULL)] = t[1] + t[2]
  v[WHERE(d gt t[0], /NULL)] = 0
  return, v
end
```

Thoughts?

-Chris
IDL Product Lead
ExelisVIS

Subject: Re: How to speed up KRIG2D by 30x
Posted by Mike[5] on Thu, 10 Oct 2013 11:44:05 GMT
View Forum Message <> Reply to Message

Hi Chris,

Thank you for your prompt response.

You are right, I did notice the covariances were not right. I rewrote them as variograms instead, like below:

```
FUNCTION Krig_sphere, rad, c      ;Return Spherical variogram Fcn
   r = rad/c[0] < 1.0
   return, c[1] + c[2] * (r*(1.5 - 0.5*r^2))
end
FUNCTION Krig_expon, rad, c       ;Return Exponential variogram  Fcn
   return, c[1] + c[2] * (1.0 - exp(-3.0*rad/c[0]))
end
```

Another small change I made was to use the two lines below instead of the double loop to fill the array coefficients

```
a[0,0] = distance_measure( transpose([[x],[y]]), /MATRIX )
a = call_function(fname, a, t)   ; Get coefficient matrix
```

On Wednesday, October 9, 2013 6:32:53 PM UTC+1, Chris Torrence wrote:

```
>  Hi Mike,
>
>
>
>  This is fantastic. I'm working on adding in your change. However, I just found a different problem. If you look at the "Krig_Sphere" function within krig2d.pro, the code doesn't match the docs. The documentation states that for spherical covariance:
>
>
>
>  C(d) = C1 - 1.5 C1 (d/A) + 0.5 C1 (d/A)^3  if d < A
>
>       = C0 + C1    if d = 0
>
>       = 0          if d > A
>
>
>
>  Note that I threw in a factor of C1 on the first line (the docs are wrong).
>
>
>
>  Here is the code:
>
>  FUNCTION Krig_sphere, d, t
>
>    r = d/t[0]
>
>    v = t[1] + t[2] * (r * (1.5 - 0.5 * r *r) > 0)
>
>    z = where(d eq 0, count)
```

>
>   if count ne 0 then v[z] = 0
>
>   return, (t[1] + t[2]) - v
>
> end
>
>
>
> We are not clipping to 0 for d > A. In fact, for d > A, the function actually starts to go back up and levels off at the constant C1. You can see this with the following plot:
>
> p = plot(krig_sphere(findgen(40), [10, 0.5, 1]))
>
>
>
> I think the code should be something more like:
>
> FUNCTION Krig_sphere, d, t
>
>   r = d/t[0]
>
>   v = t[2] * (1 - r * (1.5 - 0.5*r*r))
>
>   v[WHERE(d eq 0, /NULL)] = t[1] + t[2]
>
>   v[WHERE(d gt t[0], /NULL)] = 0
>
>   return, v
>
> end
>
>
>
> Thoughts?
>
>
>
> -Chris
>
> IDL Product Lead
>
> ExelisVIS

Subject: Re: How to speed up KRIG2D by 30x
Posted by chris_torrence@NOSPAM on Thu, 10 Oct 2013 16:28:27 GMT

On Thursday, October 10, 2013 5:44:05 AM UTC-6, Mike wrote:
> Hi Chris,
>
>
>
> Thank you for your prompt response.
>
>
>
> You are right, I did notice the covariances were not right. I rewrote them as variograms instead, like below:
>
>
>
```
> FUNCTION Krig_sphere, rad, c      ;Return Spherical variogram Fcn
>
>     r = rad/c[0] < 1.0
>
>     return, c[1] + c[2] * (r*(1.5 - 0.5*r^2))
>
> end
>
> FUNCTION Krig_expon, rad, c       ;Return Exponential variogram  Fcn
>
>     return, c[1] + c[2] * (1.0 - exp(-3.0*rad/c[0]))
>
> end
```
>
>
>
> Another small change I made was to use the two lines below instead of the double loop to fill the array coefficients
>
>
>
```
> a[0,0] = distance_measure( transpose([[x],[y]]), /MATRIX )
>
> a = call_function(fname, a, t)   ; Get coefficient matrix
```
>
>

Hey Mike,
So when you are using the variograms, what does the rest of the code look like? I just want to make sure that we get the same answer.
-Chris

Subject: Re: How to speed up KRIG2D by 30x
Posted by chris_torrence@NOSPAM on Thu, 10 Oct 2013 17:48:56 GMT
View Forum Message <> Reply to Message

Okay, here is the new code, minus the comments. I added 3 new keywords: DOUBLE, XOUT, YOUT. DOUBLE forces double precision. XOUT, YOUT are output keywords that contain the final X/Y locations. I also vectorized the final loop, for even more speed.

I also changed the C code of GRIDDATA to move the LUSOL out of the loop.

Now, with Mike's changes and the vectorization, I'm getting the following results using Mike's test code at the top with 500 input points and 400 output points:

Old KRIG2D: 30 seconds
New KRIG2D: 0.51 seconds
New GRIDDATA: 1.3 seconds

It's funny that the KRIG2D is faster than GRIDDATA, but that's because GRIDDATA has a lot of other machinery to handle sectors & search ellipses. In fact, if you *do* use sectors or search_ellipse, then GRIDDATA has to revert to the old slow algorithm with the ludc in the inner loop. Oh well.

Anyway, let me know how this code looks. If all goes well, this will make it into IDL 8.3, due out in a month or so.

Mike, thanks again for the patch to the code.

Cheers,
Chris
ExelisVIS

```
; Copyright (c) 1993-2013, Exelis Visual Information Solutions, Inc. All
;       rights reserved. Unauthorized reproduction is prohibited.
;Return Exponential Covariance Fcn
; C(d) = C1 exp(-3 d/A)   if d != 0
;      = C1 + C0  if d == 0
;
FUNCTION Krig_expon, d, t
  COMPILE_OPT idl2, hidden

  r = t[2] * exp((-3./t[0]) * d)
  r[WHERE(d eq 0, /NULL)] = t[1] + t[2]
  return, r
end

;Return Spherical Covariance Fcn
; C(d) = C1 [ 1 - 1.5(d/A) + 0.5(d/A)^3] if d < A
;      = C1 + C0  if d == 0
;      = 0        if d > A
```

```
;
FUNCTION Krig_sphere, d, t
  COMPILE_OPT idl2, hidden

  ; The clipping to < 1 will handle the case d > A, so that v = 0.
  r = d/t[0] < 1
  v = t[2]*(1 - r*(1.5 - 0.5*r*r))
  v[WHERE(r eq 0, /NULL)] = t[1] + t[2]
  return, v
end


;   CT, VIS, Oct 2013: Per Mike's suggestion, speed up the algorithm by
;       moving LUSOL out of the loops, and vectorizing the inner loop.
;       Add DOUBLE, XOUT, YOUT keywords.
;       Fix the spherical covariance computation for d > A.
;
;-
FUNCTION krig2d, z, x, y, $
  DOUBLE=doubleIn, $
  REGULAR = regular, XGRID=xgrid, $
  XVALUES = xvalues, YGRID = ygrid, YVALUES = yvalues, $
  GS = gs, BOUNDS = bounds, NX = nx0, NY = ny0, $
  EXPONENTIAL = exponential, SPHERICAL = spherical, $
  XOUT=xout, YOUT=yout

  compile_opt idl2, hidden
  on_error, 2

  s = size(z)   ;Assume 2D
  nx = s[1]
  ny = s[2]

  reg = keyword_set(regular) or (n_params() eq 1)
  dbl = KEYWORD_SET(doubleIn)

  if n_elements(xgrid) eq 2 then begin
    x = (dbl ? dindgen(nx) : findgen(nx)) * xgrid[1] + xgrid[0]
    reg = 1
  endif else if n_elements(xvalues) gt 0 then begin
    if n_elements(xvalues) ne nx then $
      message,'Xvalues must have '+string(nx)+' elements.'
    x = xvalues
    reg = 1
  endif

  if n_elements(ygrid) eq 2 then begin
    y = (dbl ? dindgen(ny) : findgen(ny)) * ygrid[1] + ygrid[0]
    reg = 1
  endif else if n_elements(yvalues) gt 0 then begin
    if n_elements(yvalues) ne ny then $
      message,'Yvalues must have '+string(ny)+' elements.'
    y = yvalues
    reg = 1
  endif

  if reg then begin
    if s[0] ne 2 then message,'Z array must be 2D for regular grids'
    if n_elements(x) ne nx then x = findgen(nx)
    if n_elements(y) ne ny then y = findgen(ny)
    x = x # replicate(dbl ? 1d : 1., ny)  ;Expand to full arrays.
    y = replicate(dbl ? 1d : 1.,nx) # y
  endif

  n = n_elements(x)
  if n ne n_elements(y) or n ne n_elements(z) then $
    message,'x, y, and z must have same number of elements.'

  if keyword_set(exponential) then begin      ;Get model params
    t = exponential
    fname = 'KRIG_EXPON'
  endif else if keyword_set(spherical) then begin
    t = spherical
    fname = 'KRIG_SPHERE'
  endif else MESSAGE,'Either EXPONENTIAL or SPHERICAL model must be selected.'

  if n_elements(t) eq 2 then begin    ;Default value for variance?
    mz = total(z) / n   ;Mean of z
    var = total((z - mz)^2, DOUBLE=dbl)/n ;Variance of Z
    t = [t, var-t[1]] ;Default value for C1
  endif

  m = n + 1     ;# of eqns to solve
  a = dbl ? dblarr(m, m) : fltarr(m, m)

  ; Construct the symmetric distance matrix.
  for i=0, n-2 do begin
    j = [i:n-1]  ; do the columns all at once
    d = (x[i]-x[j])^2 + (y[i]-y[j])^2  ;Distance squared
    ;symmetric
    a[i,j] = d
    a[j,i] = d
  endfor

  a = call_function(fname, sqrt(a), t)        ;Get coefficient matrix
  a[n,*] = 1.0            ;Fill edges
```

```
  a[*,n] = 1.0
  a[n,n] = 0.0

  LUDC, a, indx, DOUBLE=dbl             ;Solution using LU decomposition

  if n_elements(nx0) le 0 then nx0 = 26 ;Defaults for nx and ny
  if n_elements(ny0) le 0 then ny0 = 26

  xmin = min(x, max = xmax)   ;Make the grid...
  ymin = min(y, max = ymax)

  if n_elements(bounds) lt 4 then bounds = [xmin, ymin, xmax, ymax]

  if n_elements(gs) lt 2 then $   ;Compute grid spacing from bounds
    gs = [(bounds[2]-bounds[0])/(nx0-1.), (bounds[3]-bounds[1])/(ny0-1.)]

  ; Subtract off a fudge factor to lessen roundoff errors.
  nx = ceil((bounds[2]-bounds[0])/gs[0] - 1e-5)+1 ;# of elements
  ny = ceil((bounds[3]-bounds[1])/gs[1] - 1e-5)+1

  xout = bounds[0] + gs[0]*(dbl ? DINDGEN(nx) : FINDGEN(nx))
  yout = bounds[1] + gs[1]*(dbl ? DINDGEN(ny) : FINDGEN(ny))

  ; One extra for Lagrange constraint
  d = dbl ? DBLARR(m) : FLTARR(m)
  result = dbl ? DBLARR(nx,ny,/nozero) : FLTARR(nx,ny,/nozero)

  az = LUSOL(a, indx, [REFORM(z,N_ELEMENTS(z)),0.0], DOUBLE=dbl)
  az = REBIN(TRANSPOSE(az), nx, n+1)
  xx = (SIZE(x, /N_DIM) eq 2) ? x : REBIN(TRANSPOSE(x), nx, n)
  dxsquare = (xx - REBIN(xout, nx, n))^2
  yy = (SIZE(y, /N_DIM) eq 2) ? y : REBIN(TRANSPOSE(y), nx, n)

  ; Do each row separately
  for j=0,ny-1 do begin
    y0 = yout[j]
    ; Do all of the columns in parallel
    d = sqrt(dxsquare + (yy - y0)^2)
    d = CALL_FUNCTION(fname, d, t)
    ; Be sure to add the last row of AZ, which is the Lagrange constraint
    result[*,j] = TOTAL(d*az, 2) + az[*,n]
  endfor

  return, result
end
```

Subject: Re: How to speed up KRIG2D by 30x
Posted by David Fanning on Thu, 10 Oct 2013 17:56:49 GMT
View Forum Message <> Reply to Message

Chris Torrence writes:

> Okay, here is the new code, minus the comments.

Chris, when I try to compile this code in IDL 8.2.3 I get this:

```
IDL> .compile -v 'krig2d.pro'
% Compiled module: KRIG_EXPON.
% Compiled module: KRIG_SPHERE.

       j = [i:n-1]  ; do the columns all at once
             ^
% Syntax error.
  At: C:\IDL\krig2d.pro, Line 106
```

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: How to speed up KRIG2D by 30x
Posted by David Fanning on Thu, 10 Oct 2013 18:27:51 GMT
View Forum Message <> Reply to Message

Chris Torrence writes:

> Anyway, let me know how this code looks. If all goes well, this will make it into IDL 8.3, due out in a month or so.

Substituting this:

```
   j = Lindgen((n-1) - i) + i
```

For this (which doesn't compile):

```
   j=[i:n-1]
```

I find that the new version runs about 80 times faster than the old versioin. But, I also find that the results are different:

```
   IDL> minmax, z1 ; Old version
   MinMax:     -0.139365      4.87036
   IDL> minmax, z2 ; New version
   MinMax:      0.954356      5.97115
```

Any ideas on how to account for this? Displaying the two arrays as
images side-by-side shows the differences.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: How to speed up KRIG2D by 30x
Posted by chris_torrence@NOSPAM on Thu, 10 Oct 2013 20:04:24 GMT
View Forum Message <> Reply to Message

On Thursday, October 10, 2013 12:27:51 PM UTC-6, David Fanning wrote:
> Chris Torrence writes:
>
>
>
>> Anyway, let me know how this code looks. If all goes well, this will make it into IDL 8.3, due out in a month or so.
>
>
>
> Substituting this:
>
>
>
>   j = Lindgen((n-1) - i) + i
>
>
>
> For this (which doesn't compile):
>
>
>

```
>    j=[i:n-1]
>
>
>
>  I find that the new version runs about 80 times faster than the old
>
>  versioin. But, I also find that the results are different:
>
>
>
>    IDL> minmax, z1 ; Old version
>
>    MinMax:     -0.139365      4.87036
>
>    IDL> minmax, z2 ; New version
>
>    MinMax:      0.954356      5.97115
>
>
>
>  Any ideas on how to account for this? Displaying the two arrays as
>
>  images side-by-side shows the differences.
>
>
>
>  Cheers,
>
>
>
>  David
>
>
>
>
>
>  --
>
>  David Fanning, Ph.D.
>
>  Fanning Software Consulting, Inc.
>
>  Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
>
>  Sepore ma de ni thue. ("Perhaps thou speakest truth.")
```

Hi David,
Thanks for trying out the code. Whoops! That [i:n-1] is a new IDL 8.3 feature. :-)

But I think your replacement code should be:
j = LINDGEN(n-i) + i

That probably explains the difference. When I compare the old to the new, I see differences of 10^-5 or less.

-Chris

Subject: Re: How to speed up KRIG2D by 30x
Posted by David Fanning on Thu, 10 Oct 2013 20:09:56 GMT
View Forum Message <> Reply to Message

Chris Torrence writes:

> Thanks for trying out the code. Whoops! That [i:n-1] is a new IDL 8.3 feature. :-)
>
> But I think your replacement code should be:
> j = LINDGEN(n-i) + i
>
> That probably explains the difference. When I compare the old to the new, I see differences of 10^-5 or less.

Ah, OK. Works great now. VERY fast! :-)

Thanks,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: How to speed up KRIG2D by 30x
Posted by Mike[5] on Fri, 11 Oct 2013 11:11:45 GMT
View Forum Message <> Reply to Message

Hi Chris,

Great job! I confirm that my modified routine using variograms gives identical results to yours within numerical accuracy, as long as I set my variograms=0 when d=0 in agreement with your covariances. This is of course after translating the interesting new IDL 8.3 feature that David spotted. Note that one can use either variograms functions or covariances without further changes to the program.

The new covariance functions now looks right and corrects for some major numerical differences one can see against the old KRIG2D, when the data extend beyond the region where the covariance was supposed to flatten out.

I like your optimizations in the final loop and the XOUT and YOUT keywords addition. And I am glad you also already modified GRIDDATA. This was an amazing efficient reaction on your side!

Cheers,
   Michele

On Thursday, 10 October 2013 18:48:56 UTC+1, Chris Torrence wrote:
> Okay, here is the new code, minus the comments. I added 3 new keywords: DOUBLE, XOUT, YOUT. DOUBLE forces double precision. XOUT, YOUT are output keywords that contain the final X/Y locations. I also vectorized the final loop, for even more speed.
>
>
>
> I also changed the C code of GRIDDATA to move the LUSOL out of the loop.
>
>
>
> Now, with Mike's changes and the vectorization, I'm getting the following results using Mike's test code at the top with 500 input points and 400 output points:
>
>
>
> Old KRIG2D: 30 seconds
>
> New KRIG2D: 0.51 seconds
>
> New GRIDDATA: 1.3 seconds
>
>
>
> It's funny that the KRIG2D is faster than GRIDDATA, but that's because GRIDDATA has a lot of other machinery to handle sectors & search ellipses. In fact, if you *do* use sectors or search_ellipse, then GRIDDATA has to revert to the old slow algorithm with the ludc in the inner loop. Oh well.
>
>
>
> Anyway, let me know how this code looks. If all goes well, this will make it into IDL 8.3, due out in a month or so.

>
>
>
> Mike, thanks again for the patch to the code.
>
>
>
> Cheers,
>
> Chris
>
> ExelisVIS

Subject: Re: How to speed up KRIG2D by 30x
Posted by David Fanning on Sat, 12 Oct 2013 15:28:39 GMT
View Forum Message <> Reply to Message

Chris Torrence writes:

> Okay, here is the new code, minus the comments. I added 3 new keywords: DOUBLE, XOUT, YOUT. DOUBLE forces double precision. XOUT, YOUT are output keywords that contain the final X/Y locations. I also vectorized the final loop, for even more speed.
>
> I also changed the C code of GRIDDATA to move the LUSOL out of the loop.
>
> Now, with Mike's changes and the vectorization, I'm getting the following results using Mike's test code at the top with 500 input points and 400 output points:
>
> Old KRIG2D: 30 seconds
> New KRIG2D: 0.51 seconds
> New GRIDDATA: 1.3 seconds

Fast Kriging may have saved the day in a map warping problem I've been
working on. This is really, really great! Thanks for being all over
this, and thanks to Michele for solving the original problem.

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: How to speed up KRIG2D by 30x
Posted by timothyja123 on Mon, 14 Oct 2013 23:47:51 GMT
View Forum Message <> Reply to Message

On Sunday, October 13, 2013 2:28:39 AM UTC+11, David Fanning wrote:
> Fast Kriging may have saved the day in a map warping problem I've been
>
> working on. This is really, really great! Thanks for being all over
>
> this, and thanks to Michele for solving the original problem.

Thanks to the power of Open Source.

Subject: Re: How to speed up KRIG2D by 30x
Posted by David Fanning on Tue, 15 Oct 2013 00:24:39 GMT
View Forum Message <> Reply to Message

timothyja123@gmail.com writes:

> Thanks to the power of Open Source.

As opposed to the confusion of Open Source, I presume you mean. But,
yes, in this case, it worked out great. ;-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")