Arrays as arguments in first-class functions: the Levenberg-Marquardt algorithm in Chapel

Nelson Luís Dias[1], Débora Roberti[2], and Vanessa Arruda[2]

[1]Department of Environmental Engineering, Federal University of Paraná, Brazil

: nelsonluisdias@gmail.com : www.nldias.github.io

[2]Department of Physics, Federal University of Santa Maria, Rio Grande do Sul, Brazil

ChapelCon '24, June 7 2024

Contents

Motivation

Curve fitting and beyond

The need:

- In Science and Engineering, it is often needed to fit a particular mathematical expression to observed data.
- This is usually done via least-squares and optimization to find the best parameters for the expression.
- The Levenberg-Marquardt (LM) Method (Levenberg, 1944; Marquardt, 1963; Fletcher, 1971) is the undisputed choice.

Some LM tools:

- Gnuplot: works well with relatively simple expressions; see `www.gnuplot.info`.
- Numerical Recipes (Press et al., 1992): works, but not very much up-to-date; see `https://www.stat.uchicago.edu/~lekheng/courses/302/wnnr/nr.html`.
- Gnu Scientific Library: up-to-date, thorough, in C, and with a steep learning curve.
- CMinpack at `http://devernay.github.io/cminpack` also in C.
- Python + SciPy: easy (it's Python!); see for example `https://hernandis.me/2020/04/05/three-examples-of-nonlinear-least-squares-fitting-in-python-with-scipy.html`.

In all cases except Gnuplot, the LM function calls a function with array arguments

Numerical Recipes:

```
void mrqmin(float x[], float y[], float sig[], int ndata,
float a[], int ia[], int ma, float **covar, float **alpha, float
*chisq,
void (*funcs)(float, float [], float *, float [], int), float *alamda)
```

CMinpack:

```
void lmdif1_ ( void (*fcn)(int *m, int *n, double *x, double *fvec, int *iflag),
int *m, int * n, double *x, double *fvec,
double *tol, int *info, int *iwa, double *wa, int *lwa)
```

...But no (known to myself) native Chapel implementation

Limitations:

- LM is not very simple to understand/implement.
- All implementations have a procedural argument that in turn has an array as an argument.
- But Chapel can only have procedures as arguments if they are ***first-class functions*** (or ***first-class procedures***).

⇒ ***A LM procedure with a relatively simple interface in Chapel would be desirable, and would streamline a Chapel-based workflow.***

A Reviewer's caveat:

> That said, I would suggest that the authors mention that there is another type of interface which implements a finite state machine for LM which better suits experienced users of the case where the evaluation of the function is highly complex.

However, I was unable to find finite state machines & LM references.

First-class procedures

Definition

From https://chapel-lang.org/docs/technotes/firstClassProcedures.html

First-class procedures can be captured as values:

```
proc myfunc(x:int) { return x + 1; }
const p = myfunc;
writeln(p(3));  // outputs: 4
```

A first-class procedure ***cannot***:

- Refer to any outer variable that is not at module scope
- Have a type or param return type
- Accept type or param formals
- Be a method
- Be overloaded
- ***Be generic***
- Be parenless

Problem: a function with an “open” array argument is generic

```
proc g(a: [] real): real {    // calculates the sum of a
   var s = 0.0;
   for e in a do {
      s += e ;
   }
   return s;
}
proc f(ref a: [] real, const ref g: proc(x: [] real)) {     // tells sign of sum
   if g(a) > 0.0 then {
      writeln("sum␣is␣positive");
   }
   else if g(a) ==  0.0 then {
      writeln("sum␣is␣zero");
   }
   else {
      writeln("sum␣is␣negative");
   }
}
var a = [1.0,2.0,-3.0];
f(a,g);
```

```
fof-fail.chpl:20:  error:  the proc 'g' is generic and cannot be captured
```

Solution: attached domain arrays

- A feature that exists in Chapel. Please see **The `this` Accessor**, in `https://chapel-lang.org/docs/primers/Methods.html`.
- Inspired by `https://stackoverflow.com/questions/48086588/how-to-create-a-ragged-array-in-chapel`, which discusses how to create ragged arrays in Chapel.

Here is a very simple implementation:

```
record vec {
   var dom: domain(1);
   var arr: [dom] real;
   proc ref this(k:int) ref {
      return arr[k];
   }
}
```

- A variable of type `vec` contains a domain and a 1-D array over this domain.
- ***A procedure with an argument of type `vec` no longer is generic — why?***

The modified program using type vec

```
proc g(ref a: vec): real {    // calculates the sum of a
   var s = 0.0;
   for i in a.dom do {
      s += a[i];
   }
   return s;
}
proc f(ref a: vec, const ref g: proc(ref x: vec)) {    // tells sign of sum
   if g(a) > 0.0 then {
      writeln("sum␣is␣positive");
   }
   else if g(a) ==  0.0 then {
      writeln("sum␣is␣zero");
   }
   else {
      writeln("sum␣is␣negative");
   }
}
var a = new vec({1..3},[1.0,2.0,-3.0]);
f(a,g);
```

The real `ada.chpl` ...

…is available at `https://nldias.github.io/software.html`

I have implemented a slightly more capable `ada` (for **a**ttached **d**omain **a**rrays) module. Main highlights:

- Two record types: `vec` for 1D attached arrays, and `mat` for 2D attached arrays.
- A `size` method.
- A limited `reindex` method.
- Overloaded arithmetic operators between **`real`** and `vec`, and **`real`** and `mat`.
- Overloaded arithmetic operators between `vec` and `vec`, and `mat` and `mat`.
- ***No*** slicing of `vec`*s* and `mat`*s*.
- ***No*** overloaded operators between arrays and `vec`*s* or `mat`*s* (frankly, things got complicated and I couldn't do it, although it seems possible).
- (Probably inefficient) `tovec` and `tomat` procedures to convert (by fully copying) arrays to `vec`*s* and `mat`*s*.

There is probably room for improving `ada.chpl`! At this point, I valued simplicity over efficiency.

A Chapel implementation of Levenberg-Marquardt

General remarks

The procedure, `levmar`, is in module `nstat.chpl`, also at https://nldias.github.io/software.html

- Based (with several adaptations) on the excellent presentation of the LM method by Gavin (2022) and its MatLab implementation.
- Uses a module `smatrix.chpl` to calculate products between matrices and vectors, solve systems of linear equations, etc.. `smatrix.chpl`'s routines are very straightforward and are ***not*** optimized: far and away from `blas`!

levmar's interface

```
// -------------------------------------------------------------
// --> levmar: nonlinear least squares by curve fitting with the
// Levenberg-Marquard method. Here x is a mat.
// -------------------------------------------------------------
proc levmar(
   ref x: mat,              // independent variables (used as arg to func) (m x ell)
   ref y: vec,              // data to be fit by func (m x 1)
   ref w: [] real,          // array, *not matrix*, of weights (m x 1)
   ref p: vec,              // initial guess of parameter values  (n x 1)
                            // returns the estimated parameters
   ref sigp: [] real,       // standard  errors of the parameters (n x 1)
   ref cp: [] real,         // parameter covariance matrix (n x n)
   const ref func: proc(ref ax: mat,    // the independent variables
                 ref ap: vec,           // the parameters
                 ref yhat: vec),        // in the sim model call func(ax,ap,yhat)
   const in epsilon_p = 1.0e-6          // stop criterion
   ) : (real,real,real)  // (red chi sq, st err of estimate, coeff det)
   where ( w.rank == 1 && sigp.rank == 1 && cp.rank == 2) {
```

Simple arithmetic with vecs

```
proc simplejacob() {
   const delp: [1..n] real = 1.0e-6;
   var forwp = new vec({1..n});
   var backp = new vec({1..n});
   var yplus = new vec({1..m});
   var yminus = new vec({1..m});
   for k in 1..n do {
      forwp = p;
      backp = p;
      forwp[k] += delp[k];
      backp[k] -= delp[k];
      func(x,forwp,yplus);
      func(x,backp,yminus);
      J[1..m,k] = (yplus.arr[1..m] - yminus.arr[1..m])/(2*delp[k]);
   }
}
```

Applications with meteorological data

A model for atmospheric radiation

Daily data: R_s (measured solar radiation), R_{sea} (calculated solar radiation at the top of the atmosphere), e_a (measured water vapor pressure), T_a (measured air temperature)

$$S = (1/b_P)(R_s/R_{sea} - a_P);$$
$$C = 1 - S;$$
$$R_{ac} = a_B(e_a/T_a)^{b_B}\sigma T_a^4;$$
$$R_a = \left(1 + c_B C^{d_B}\right) R_{ac}.$$

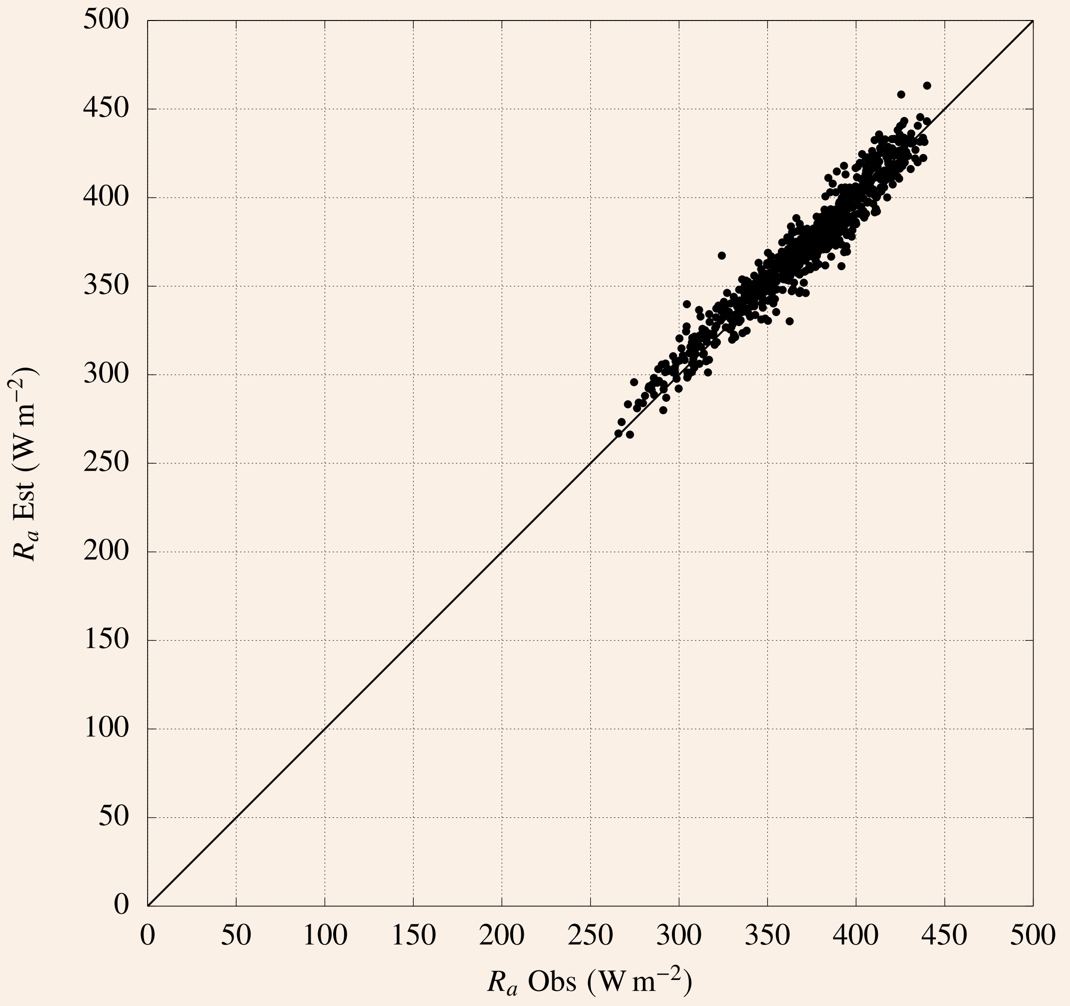
$\sigma = 5.670374419 \times 10^{-8}\,\mathrm{W\,m^{-2}\,K^{-4}}$ (Stefan-Boltzmann constant)

6 parameters to estimate: $(a_P, b_P, a_B, b_B, c_B, d_B)$

Data measured over a rice paddy in Rio Grande Sul state, Brazil.

x is a matrix of 725 days × 4 values of (R_s, R_{sea}, e_a, T_a)

y is a vector of 725 values of measured R_a

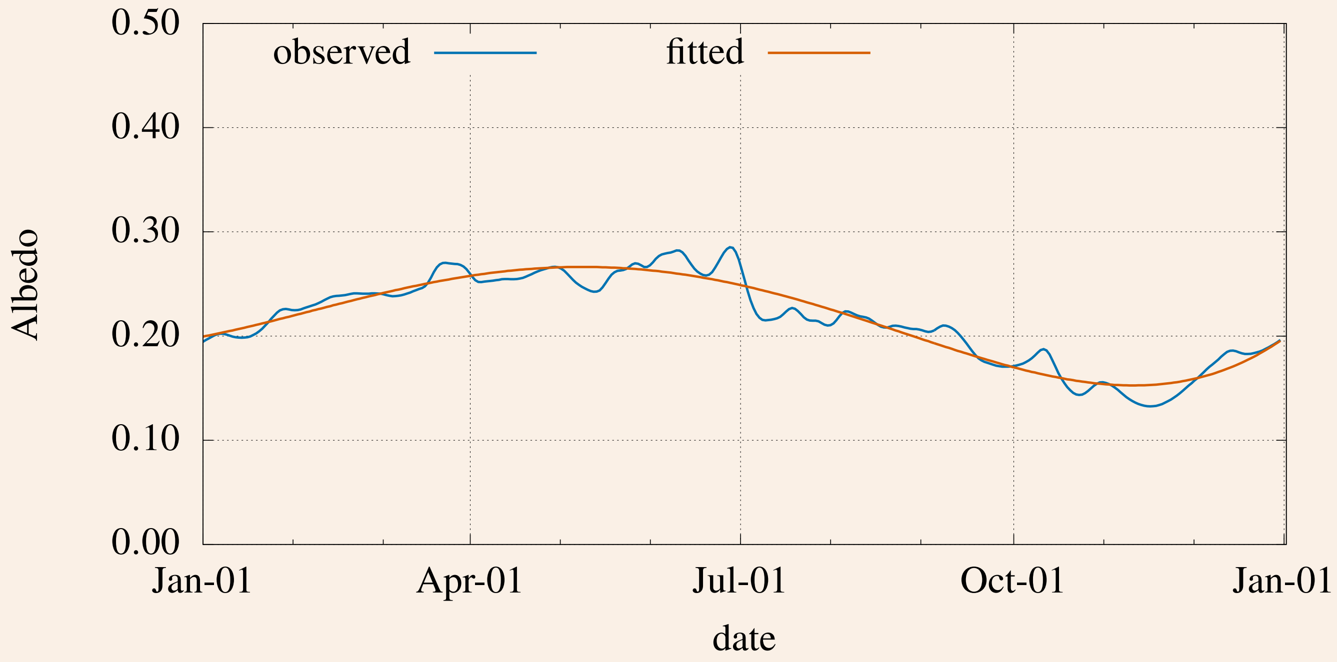


A polynomial fit for the seasonal variation of the albedo

Daily data: measured albedoes (reflected/incoming solar radiation)
Adjust a 4th-degree polynomial to measured albedo,

$$\hat{y} = p_0 + p_1 x + p_2 x^2 + p_3 x^3 + p_4 x^4,$$

where x is the day of the year, between 1 and 365.
x is a matrix of 365 days × 1 value of albedo (actually a vector).
y is a vector of 365 values of measured albedo.



Conclusions

Conclusions

- `vec`*s* and `mat`*s* are not generic types, and overcome the limitations imposed on '`[] real`' for 1st-class procedures.
- They are created as (for example)

```
var alb = new vec({1..10});
```

  and can be accessed element-by-element as arrays (`alb[i] = ...`), etc..
- In this talk they were indispensable to implement a practical procedure to do non-linear least squares with the Levenberg-Marquardt method.
- There is probably room for improvement both in `ada.chpl` (which implements `vec` and `mat`) and `levmar`; for now I chose simplicity of implementation over efficiency.

Thanks for the attention.

References

Fletcher, R. (1971). A modified Marquardt subroutine for non-linear least squares. Technical report, Theoretical Physics Division, Atomic Energy Research Establishment Harwell, UK.

Gavin, H. P. (2022). *The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems.* Available at https://people.duke.edu/~hpgavin/ExperimentalSystems/lm.pdf.

Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2), 164–168.

Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2), pp. 431–441. http://www.jstor.org/stable/2098941

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C; The Art of Scientific Computing* (2nd ed.). Cambridge University Press.