ncDataReader2 - User Manual

version 2.3.x

Contents

1 Overview

ncDataReader2 is a library of C functions to access data stored in netCDF files using different interpolation and extrapolation methods. The aim of this library is to provide access from simulation systems like Modelica to data sets like weather data or measured time rows. As such systems usually require strictly continuity of functions and their derivatives, smooth spline interpolation is included.

netCDF is a very efficient binary file format for structured multidimensional data. The netCDF library is freely available on all major platforms. ncDataReader2 works both with netCDF versions 3.x and 4 (which is based on HDF5).

ncDataReader2 supports reading one dimensional data (like generated or measured time rows of simple quantities), using periodic extrapolation if needed. Interpolation methods currently supported are discrete steps, linear, akima splines and smoothed steps.

Support for variables that depend on two dimensions (scattered points, lists of x,y,z-pairs) is included but not very well tested. The 2D functions use the csa library for cubic spline interpolation by Pavel Sakov.

ncDataReader2 will build as a static or dynamic library on Linux, Windows and MacOS X.

2 Author and License

ncDataReader2 was developed by Joerg Raedler (joerg@j-raedler.de). The code is released under the terms of the 'GNU Lesser General License'_. The code in the files `csa.c`, `csa.h`, `csa_config.h`, `svd.c` and `svd.h` was taken from the csa library which has its own open source license.

3 Building and Installation

The build process uses cmake to configure the sources. To compile ncDataReader2 you will need netCDF, cmake and a compiler/development system for C/C++. ncDatareader2 was tested with:

- gcc and tcc on linux platforms
- cygwin, MinGW or Microsoft Visual Studio (including the free Express Edition) on Windows platforms
- XCode developer tools on MacOS X

Use `cmake` to configure the sources and build system, then build the library and examples. On linux you use the command `make . && make` in the source folder to do this.

The installation procedure is not yet automated, you should copy the relevant files manually to the needed location. To compile your programs with ncDataReader2 you need a library file and the header file(s). Library files are:

- **Linux:**
  - `libncDataReader2.a` (static) or
  - `libncDataReader2.so` (dynamic)
- **Windows:**
  - `libncDataReader2.a` (cygwin or MinGW)
  - `ncDataReader2.dll` and `ncDataReader2.lib` (Visual Studio)
- **MacOS X:**
  - ???

The header file is called `ncDataReader2.h` for the general API and `ncDataReaderEA.h` for the easy API.

4 Examples

You will find some examples in the folder `examples`. You should run `GenerateFile(.exe)` first to create the netCDF file the other example programs will need. `GenerateBigFile(.exe)` will create a large file that is used by some of the examples.

A simple example:

```
#include "ncDataReader2.h"

NcDataSet1D *x = ncDataSet1DNew("daten.nc", "time", EpPeriodic, LtFull, 10);
NcVar1D     *y = ncVar1DNew(x, "var1", IpAkima, LtFull);

double tmp = ncVar1DGet(y, 42.0);
```

This will open the independant variable `time` and the dependant variable `var1` in a file `daten.nc`, calculate the interpolated value of `var1` for `time=42.0`. The variable `time` will be used periodic, `var1` will be interpolated by the Akima method. All data will be fully loaded.

5 Concept

A one dimensional data set (NcDataSet1D) is the representation of one independant variable in a netCDF file. This data can be equally spaced, but it doesn't need to. A one dimensional variable (NcVar1D) is the representation of a dependant variable that has a dependency to exactly one NcDataSet1D. The value of a NcVar1D at a certain point can be evaluated (usually interpolated). A NcDataSet1D can be referenced by more than one NcVar1D.

Example: a file contains weathr data as time rows (e.g. hourly values). One variable (time) contains the time values at which other quantities were measured. The other time rows (temperature, humidity, radiation) contain the measured values. With ncDataReader2 we would reference 'time' as a NcDataSet1D. 'temperature', 'humidity' and 'radiation' are referenced as single NcVar1D's which are connected to the this set. For every possible value of 'time' we can now evaluate the quantities and get (possibly interpolated) values.

You can reference the same variables in a file multiple times with different parameters as different NcDataSet1D or NcVar1D.

5.1 Interpolation

5.1.1 Discrete

This is the simplest but fastest method. The value of a variable is the value of the last data point where the value of the independant variable is smaller or exactly equal to the demanded point. This will lead to steps at the intervall boundaries. Neither the function nor the derivatives are continuous.

5.1.2 Linear

Linear interpolation between the points leads to a continuous function with non-continuous derivatives and is very fast.

5.1.3 Akima

Akima interpolation is a cubic spline interpolation method. The calculation is not fast, but the result is a very smooth function (continuous curve and derivative). The continuity of the second derivative was abandoned to get only local dependencies of the parameters. This is a big advantage in comparison to normal cubic splines wheree all values of a data set have to be taken into account to calculate a single value.

By using Akima interpolation with ncDataReader2 you can get a smooth interpolation of very large variables by reading only a few values of the required range from the file.

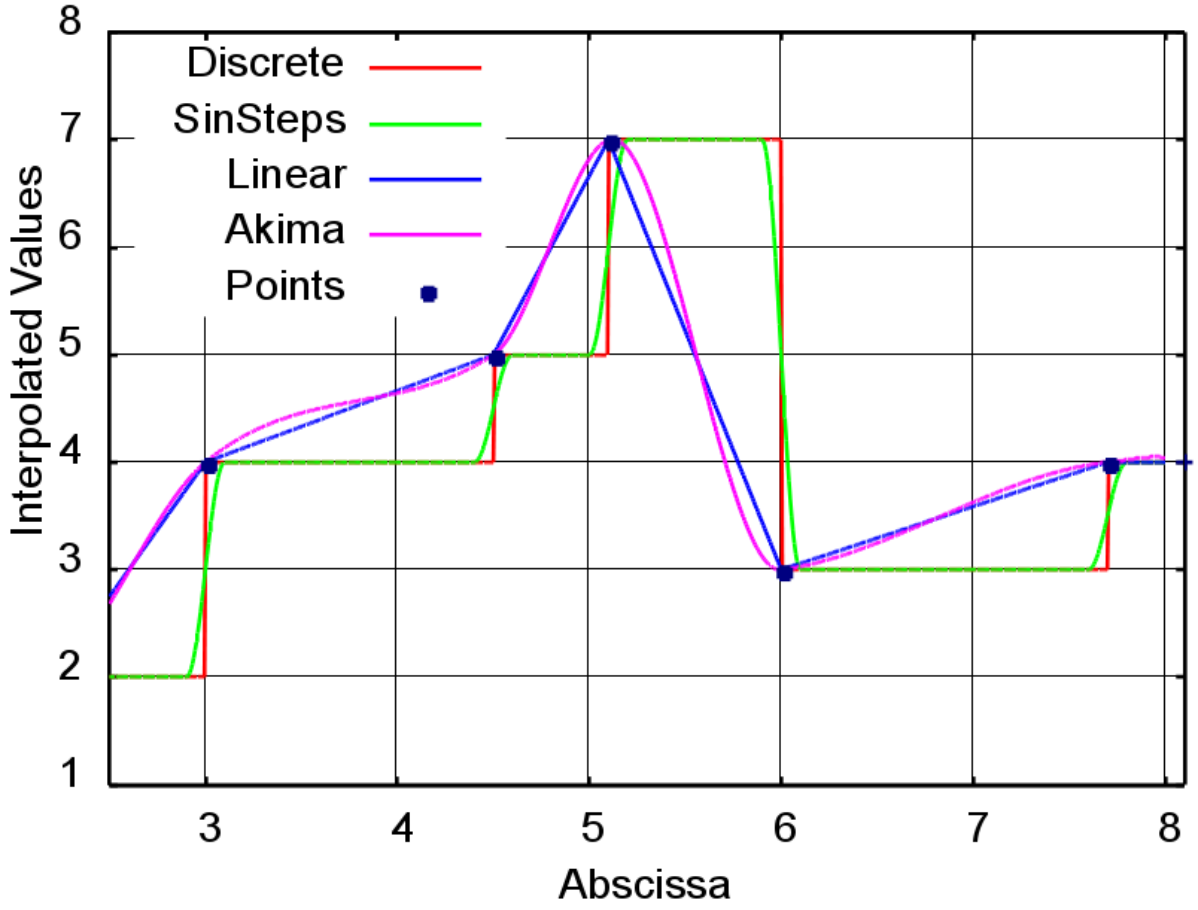
5.1.4 Sine Steps (SinSteps)

This is a variation of the discrete method where the steps are smoothed by inserting parts of the sine function. In the middle of an intervall the values are still constant (step-like behaviour) but at the intervall boundaries a smooth transition is ensured. The amount of smoothing can be configured by defined by an influence radius around the points. The resulting curve is continous and has a continuous first derivative. Strictly speaking, this is not an interpolation method since the data points are usually not met.

5.1.5 Cosine Window (CosWin)

This is method does also more an approximation than a real interpolation. It will calculate the weighted average of all points and their linear interpolation inside a window. The weighting function is cos(x) (scaled and shifted).

The result is continuous and has a continuous first derivative. For small windows the curve will follow the linear interpolation with some smoothing around the points. Large windows lead to more smoothing of the dats set. The window size should be much smaller than the data range of the abscissa.



5.2 Transformations

Every variable can be automatically shifted and scaled by setting an offset and a scale factor to avoid later conversions. This is very handy if you need to convert between different units. A NcDataSet1D can be used in periodic mode where the data virtually continues after the end or before the start. This way you may use a generated weather file with one year of data to simulate severals years with continuous time values.

5.3 Loading Data

As variables in netCDF files can be very large different methods of loading are supported. Small variables (few values) can be loaded completely into memory to get the fastest access. The other extreme is to load every single value only on demand. This is significant slower but still fast because of the very efficient netCDF file acces. A third possibility is to load chunks of data on demand.

5.4 Optimization

To optimize the calculation different caches can be activated. All caches are implemented as ring buffers with a specific capacity. With a capacity of x, the last x items are cached and can be retrieved very fast. But large capacities will lead to a large overhead and slow down the calculation. That's why the cache sizes can can't be optimized globally but should carefully be adapted to the current problem. An example program CacheTests(.exe) demonstrates the effect of the caches.

5.4.1 Value Cache

The value cache stores the calculated values of a NcVar1D for a specific value of the independant variable. DAE based simulation systems tend to call the same functions with the same arguments very often. A value cache can speed up the calculations in this case.

5.4.2 Lookup Cache

When the value of a NcVar1D is requested the first action is to search the corresponding intervall of the NcDataSet1D with a nested search over the whole data set. For large data sets this may need a lot of time. The lookup cache stores the last used intervalls and their boundaries. If the next requested value is in the same interval as the last one(s), this may speed up the search.

5.4.3 Parameter Cache

Linear and Akima interpolation methods need to calculate the parameters of a linear or a cubic function for one intervall. These parameters can be stored in a cache. If the next requested value is in the same interval as the last one(s), this may speed up the caclulation (in particular for the Akima method).

5.5 Scattered Points (2D)

Variables that depend on two dimensions are defined as a list of 3D points that can be scattered in 3D space. Those points are read from a two dimensional variable in the netCDF file. At initialization time a spline surface is constructed from the points. Some parameters (npmin, npmax, k and nppc) affect the construction and the quality of the spline surface.

After this step the interpolated value of z for arbitrary values of x and y can be calculated.

5.6 File annotations / netCDF Attributes

All the parameters like interpolation and extrapolation methods, transformations, cache sizes and other can be explicitely set or can be read from the data file. netCDF files may contain generic attributes (global and and variable specific). Some attributes with special names are honoured by ncDataReader2. Parameters set with explicit functions always have precedence over annotations. The following attributes are supported for netCDF variables:

Name	Possible Values	Meaning
scale_factor	float value	scaling factor
add_offset	float value	offset
extrapolation	"default", "periodic"	extrapolation method for indep. variables

interpolation	"discrete", "linear", "akima", "sinsteps"	interpolation method for dep. variables
load_type	"auto", "full", "none", "chunk"	data load type
chunk_size	integer value	chunk size for data loading
smoothing	float value	smoothing radius
lookup_cache	integer value	capacity of lookup cache
value_cache	integer value	capacity of value cache
parameter_cache	integer value	capacity of parameter cache
csa_npmin	integer value	npmin parameter for csa
csa_npmax	integer value	npmax parameter for csa
csa_k	integer value	k parameter for csa
csa_nppc	integer value	nppc parameter for csa

6 General API

To use the general API you should include the header file `ncDataReader2.h`. Data sets and variables are represented by C-structs. You should not try to initialize or destroy these struct objects yourself but to use the provided functions instead.

6.1 NcDataSet1D

NcDataSet1D is a struct object which holds all information on a data set. A new NcDataSet1D will be created with the following function:

```
NcDataSet1D *ncDataSet1DNew(char *fileName,
                            char *varName,
                            Extrapolation extra,
                            LoadType loadType,
                            size_t lookupCacheSize);
```

`fileName` is the name of the file, `varName` is the name of the independant variable. `extra` is the method of extrapolation (the behaviour when the defined data range is left). The variable `loadType` defines the way the data is loaded from file to memory. The parameter `lookupCacheSize` is the size of the lookup cache for this data set. Use `NC_LOOKUP_CACHE_AUTO` to read this value from the file annotation. If not set, no cache will be used.

Possible values for the extrapolation method (`Extrapolation`) are:

- `EpDefault` - use a method corresponding to the interpolation method. This is the first or last value for discrete or sine steps and the linear cubic extrapolation using the parameters of the first/last interval for linear or Akima extrapolation.
- `EpPeriodic` - adjust values for periodic use. The first and last values of the data set must mark the boundaries of the periodic range. Example: time-dependant values for one whole day should start with a value for 0:00 and end with a value for 24:00 to get a daily periodic data set. If the first and last values of a NcVar1D are not equal, they will be replaced with an average transition value.
- `EpConstant` - use the border values when outside.
- `EpAuto` - the extrapolation method will be read from the file annotation. If not set `EpDefault` will be used.

The load type (LoadType) can be one of the following:

- LtFull - the full variable will be loaded to memory.
- LtNone - every single value will be read from the file on demand.
- LtAuto - use the file annotation. If not set, LtFull will be used for small variables and LtNone for large ones. The limit is defined as LARGE_DATASET in ncDataReader2.h.
- LtChunk - load chunks of data on demand. The size can be set with an option. For a NcDataSet1D this will usually be slower than LtNone because the interval search needs the whole data range.

A NcDataSet1D should be freed with the following function when no more NcVar1D is connected to it. This will release all used memory and close netCDF objects like variables and files.:

```
void ncDataSet1DFree(NcDataSet1D *dataSet);
```

Intervall search for a value of the independant variable:

```
size_t ncDataSet1DSearch(NcDataSet1D *dataSet, double *x);
```

Get the value for one interval:

```
double ncDataSet1DGetItem(NcDataSet1D *dataSet, size_t i);
```

Set an option for a data set with this var-arg function:

```
int ncDataSet1DSetOption(NcDataSet1D *dataSet, DataSetOption option, ...);
```

Possible options are:

- OpDataSetScaling - set scaling and offset of the variable with the following two double arguments. This corresponds to the netCDF attributes scale_factor and add_offset.
- OpDataSetLookupCacheSize - change the capacity of the lookup cache to the value of the following integer value.
- OpDataSetChunkSize - change the chunk size.

6.2 NcVar1D

NcVar1D is a struct object which holds all information on a variable. For existing data set objects new variables can be defined:

```
NcVar1D *ncVar1DNew(NcDataSet1D *dataSet,
                    char *varName,
                    Interpolation inter,
                    LoadType loadType);
```

dataSet is a NcDataSet1D object, varName the name of the dependant variable in the file. You may choose the interpolation method (Interpolation) from the following values:

- IpDiscrete - discrete steps
- IpSinSteps - discrete steps with smoothing by a sine function. The smoothing radius can be defined by setting the smoothing option. If not set, a value of 0.0 will be used which will lead to the same result as IpDiscrete.

- `IpLinear` - piecewise linear Interpolation
- `IpAkima` - piecewise cubic interpolation
- `IpCosWin` - cossine window approximation. The window size can be defined by setting the `window_size` option. If not set, a value of 1.0 will be used.
- `IpAuto` - determine the interpolation method from file annotations. If not set, `IpAkima` is used.

The possible values for `LoadType` are the same as for the `NcDataSet1D`.

The calculation of values from a NcVar1D (which is the main purpose of this library) is done with the function:

```
double ncVar1DGet(NcVar1D *var, double x);
```

`var` is the NcVar1D object and x the value of the independant variable at the requested point.

To get the value of the variable (without any interpolation) in one interval you may call:

```
double ncVar1DGetItem(NcVar1D *var, size_t i);
```

A NcVar1D should be freed with the following function when it's not needed anymore. This will release all used memory and close netCDF variable object:

```
void ncVar1DFree(NcVar1D *var);
```

Set an option for a variable with this var-arg function:

```
int ncVar1DSetOption(NcVar1D *var, VarOption option, ...);
```

Possible options are:

- `OpVarScaling` - set scaling and offset with the following two double arguments. This corresponds to the netCDF attributes `scale_factor` and `add_offset`.
- `OpVarSmoothing` - set the following double value as the smoothing radius for the interpolation method `IpSinSteps`. This value has to be smaller than the smallest interval length of the data set.
- `OpVarWindowSize` - set the following double value as the window size for the interpolation method `IpCosWin`. This value should be much smaller than the data range.
- `OpVarValueCacheSize` - set the capacity of the value cache to the following integer value.
- `OpVarParameterCacheSize` - set the capacity of the parameter cache to the following integer value. This is only useful for the interpolation methods `IpLinear` and `IpAkima`.
- `OpVarChunkSize` - set the chunk size to the following integer value when using `LtChunk`.

6.3 NcScattered2D

NcScattered2D is a struct object which holds all information on a data set of scattered points and its spline interpolation. A new NcScattered2D object can be defined with:

```
NcScattered2D *ncScattered2DNew(char *fileName, char *varName);
```

`fileName` is the name of the netCDF file and `varName` the name of the variable that contains the point coordinates. `varName` should be a two dimensional variable (list of 3D points).

Befor you can request interpolated values you have to initialize the data (construct the spline surface) by calling:

```
void ncScattered2DInit(NcScattered2D *data);
```

To get an interpolated value you may call:

```
double ncScattered2DGet(NcScattered2D *data, double x, double y);
```

A NcScattered2D object should be freed after usage by calling:

```
void ncScattered2DFree(NcScattered2D *data);
```

Several options can be set by calling this var-arg function:

```
int ncScattered2DSetOption(NcScattered2D *data,
                           Scattered2DOption option,
                           ...);
```

This is call is valid only befor ncScattered2DInit() was called! Possibkle options are:

- `OpScattered2DScaling` - set scaling and offset with the following two double arguments. This corresponds to the netCDF attributes `scale_factor` and `add_offset`. This call will scale and shift all three dimensions!
- `OpScattered2DScalingX`, `OpScattered2DScalingY`, `OpScattered2DScalingZ` - set scaling and offset only in one dimension.
- `OpScattered2DPointsMin` - set the npmin parameter for csa
- `OpScattered2DPointsMax` - set the npmax parameter for csa
- `OpScattered2DPointsPerCell` - set the nppc parameter for csa
- `OpScattered2DK` - set the k parameter for csa

6.4 Error handling

netCDF functions may return errors. Errors are represented by an integer id and a message string. The default error handler will print the message to `stderr` and exit the program, on Win32 systems it will open an error dialog. You may replace this with your own handler function of the form:

```
void myhandler(int id, char *message);
```

by calling the function:

```
NcErrorHandler ncSetErrorHandler(NcErrorHandler newHandler);
```

This will set the function `newHandler` to be the new error handler and return a pointer to the previous handler.

6.5 Access Statistics

To tune the different optimization parameters some statistics can be dumped:

```
void ncDataSet1DDumpStatistics(NcDataSet1D *dataSet, FILE *f);
void ncVar1DDumpStatistics(NcVar1D *var, FILE *f);
```

This will write some statistics about the data set or the variable to a file. `f` may be a writable file pointer or NULL for stdout.

7 Easy API (EA)

The easy API was motivated by the fact that languages like Modelica cannot handle C-structs, pointers and other language elements used in ncDataReader2. They require simple functions that return values without large initializations blocks and local data storage.

The EA is a wrapper around the general API of the library that hides most of its details. To use the EA you have to include the header file `ncDataReaderEA.h`. The EA is based on hashtables that store data sets and variables after the first use. The main function is:

```
double ncEasyGet1D(char *fileName, char *varName, double x);
```

It will return the interpolated value of the variable `varName` in the netCDF file `fileName` at the point `x`. At the first call the needed NcVar1D and NcDataSet1D objects are initialized. Following calls to this function will reuse these objects. All parameters like extrapolation, interpolation, scaling, cache sizes and others are read from file annotations or set to default values.

A strict requirement to get this initialization automatically done is to follow a naming convention: the independant variable in the file must have the same name as the dimension that is used both for the independant and the dependant variable.

The functionality for 2D interpolation is also exposed via the EA:

```
double ncEasyGetScattered2D(char *fileName, char *varName,
                            double x, double y);
```

This will return the value of `z` for the position defined by `x` and `y` of a spline surface. This surface represents the list of scattered points defined by the variable `varName` in the netCDF file `fileName`. All parameters for the surface will be read from file annotations or set to default values. At the first call to this function the data is read and the surface is constructed, following calls will reuse the objects.

If you want to clean all stored objects of the EA, you may call:

```
void ncEasyFree();
```

Access statistics for all open data sets and variables can be dumped with the function:

```
int ncEasyDumpStatistics(const char *fileName);
```

There are some more functions that return attributes of the netCDF file or of variables:

```
double ncEasyGetAttributeDouble(char *fileName, char *varName, char *attName);
long   ncEasyGetAttributeLong(char *fileName, char *varName, char *attName);
char  *ncEasyGetAttributeString(char *fileName, char *varName, char *attName);
```

These functions may be used to read additional data like location coordinates for weather files. Special values will be returned on errors (like non-existent attributes), defined as `NC_DOUBLE_NOVAL`, `NC_LONG_NOVAL` and `NC_STRING_NOVAL`. If `varName` is an empty string (`""`) the global attribute is returned.

8 Modelica Interface

A modelica package is included in the folder Modelica. It contains function wrappers for the Easy API as well as some examples.

9 Preparing netCDF Files

[... this gap needs to be filled ...]

10 Tips and Tricks

[... this gap needs to be filled ...]

11 Changes

2.3.0

- added CosWin approximation
- added constant extrapolation
- GenerateBigFile is much faster now
- fixed a bug with Akima and default extrapolation near the right border
- added functions to dump statistics
- added error dialog for Win32