Putting on a Good Show with HDF5, ILNumerics, and PowerShell

Gerd Heber, The HDF Group <gheber@hdfgroup.org>

Haymo Kutschbach, ILNumerics <h.kutschbach@ilnumerics.net>

Abstract

In this article, we describe how a simple combination of three freely available tools — HDF5, ILNumerics, and PowerShell — can be used to analyze and visualize data stored in HDF5 files (on Windows). There are other excellent free choices that achieve the same (or more), but this particular combination brings together an unexpected cast of characters. This article should be of interest to HDF5 users on Windows, PowerShell fans, and people who have doubts about the utility of managed runtimes (such as .NET) for numerical calculations and visualization.

> *Thunder and lightning. Enter three Witches.*
>
> 1. When shall we three meet againe? In Thunder, Lightning, or in Raine?
> 2. When the Hurley burley's done, When the Battaile's lost, and wonne.
> 3. That will be ere the set of Sunne.
>
> —Shakespeare, The Tragedie of Macbeth.

1. Dramatis Personae

HDF5

HDF5 files [HDF5] are a common way of storing self-describing webs of numerical and non-numerical (text, multimedia) datasets of all sizes. They are popular, for example, in the High-Performance Computing [VPIC] and the Remote Sensing communities. [HDF-EOS] It is common to access HDF5 files or stores via some kind of API. The APIs can be as low-level as a C- or FORTRAN-API, or they may come with high-level conveniences such as `h5py` [h5py] or MATLAB.

ILNumerics

ILNumerics [ILNumerics] brings powerful numerical computing and visualization capabilities to the .NET framework. It allows for a convenient formulation of even complex algorithms with the help of n-dimensional array classes and a syntax akin to MATLAB or IDL. By paying attention to certain features of managed languages, algorithms implemented with ILNumerics execute at speeds competitive with C and FORTRAN.

The library has been around since 2006 and proven its efficiency for numerical algorithms and visualization in all areas where the .NET framework is commonly used: enterprise applications, ASP.NET websites, office applications and more.

PowerShell

Unbeknownst to many, Windows PowerShell [Payette2011] has been around for almost 6 years and Team PowerShell just released version 3 of the product. [WMF3] After years of overlapping,

incomplete, and incoherent Windows scripting solutions, PowerShell has emerged as the de facto standard for automation on the Windows platform. Today, no server product (e.g., Exchange, SQL Server, Hyper-V) ships without a PowerShell interface.

The great acceptance of PowerShell among administrators may have contributed to the common but ill-conceived notion that PowerShell is for administrators only. There's increasing awareness [Finke2012] that PowerShell is a great tool for developers and other keyboard users alike.

2. Pulling it off

As stated in the abstract, we would like to analyze and visualize data stored in HDF5 files. We need to connect to the bytes on disk in an HDF5 file and, on the other end, we need to connect to some kind of rendering engine that puts colorful pixels on a screen. A PowerShell module for HDF5, such as PSH5X [PSH5X], can take care of the first part. It delivers HDF5 dataset or attribute values as .NET arrays and we could then consider writing the code to render the right GUI controls, e.g., WinForms. For just a simple surface plot this doesn't sound like an attractive proposition; way too much coding! In addition, if the data need to be filtered or transformed in any substantial way, plain .NET arrays aren't exactly a great starting point.

What we need is that magic `Numpy`-like [Numpy] sauce, with convenient and efficient array manipulation. Add to that a standard portfolio of visualization primitives (plots, graphs, scenes, lights, materials, etc.) and the work will almost take care of itself. It's not a dream! It's called *ILNumerics*, and we kick off this section with a few technical comments on ILNumerics.

2.1. Why ILNumerics is so fast

Two important goals were driving the design of ILNumerics: convenience and performance. The library utilizes a whole series of means to archieve optimal execution speed. Some are potentially useful and common to all numerical applications, and others deal with the particularities of the managed framework.

2.1.1. Memory Management

It is well known that over the last roughly 30 years DRAM latencies have not kept pace with microprocessor speeds [HennPatt2012]. The 'Locality Principle' led to the introduction of a multi-level hierarchy of caches which improves the situation mainly for common business scenarios, i.e., smaller to mid-sized data schemes. Special cache-aware algorithms are needed to take advantage of caches for BigData applications. ILNumerics supports the 'Locality Principle' by forcing all arrays to always reuse the system memory [HPCWith.NET].

The library comes with a memory pool from which the storage for all computational arrays is requested. The pool takes care of the optimal mapping from available memory to incoming requests, as well as controlling its own size. In order to always have a matching array available, all unreferenced arrays dispose of their storage into the pool instead of returning it to the virtual memory manager. This is usally the job of the garbage collector (GC). However, ILNumerics takes a different approach.

While the Common Language Runtime (CLR) garbage collector does an excellent job cleaning up regular business objects, the garbage left behind by numerical algorithms is different. In fact, overall performance would drop significantly if we were to rely on the GC for larger arrays, because of highly counterproductive de-/allocation patterns. Thats why in ILNumerics arrays are disposed of *deterministically*. How is this achieved?

Since memory management a la `free` or `delete` is not supported in .NET, a combination of RIIA principles [RIIA] and artificial scoping is used. The class design for ILNumerics arrays supports this scheme by implementing several distinct array types:

- `ILArray<T>` - local arrays, valid for the current scope block

- `ILInArray<T>` - input parameter arrays, only valid in function declarations
- `ILOutArray<T>` - input/output (reference) parameter arrays, only valid in function declarations
- `ILRetArray<T>` - return type for *all* numerical functions

In implementations of algorithms, all array variables declared in a function body are local arrays (`ILArray<T>`). In order to dispose of a local array, artificial scopes are created with the help of `using` blocks (C#) (see [using Statement]). Usually such a `using` block simply encloses the whole function body. Once the execution leaves a block scope (planned or as the result of an exception), ILNumerics disposes of all arrays created within the block automatically, thus replenishing the memory pool.

The memory pool is even greedier; it asks for the *complete* set of memory used by an implementation and it wants that memory as soon as possible. In order to achieve that another simple rule acts under the hood: whenever an array is created in ILNumerics it is of type `ILRetArray<T>`. (All functions, indexers and properties in ILNumerics return objects of that type.) `ILRetArray<T>`, however, is volatile by design and it disposes of itself *after the first use*. Therefore, if a temporary arrays is not converted to a local array, it will immediately be disposed of and the memory cycle is closed.

If this sounds rather complicated, don't worry! The details are well hidden from the user. Follow three simple rules (see Appendix A, *ILNumerics Rules*) and ILNumerics will save you a bundle in memory and plenty of execution time! And there are a few other goodies that come along with that. (Read on!)

2.1.2. It ain't over 'til it's over

While efficient memory management and use have the biggest impact on execution speed, some additional features help optimize the performance even further:

- Value semantics for function parameters is achieved due to the immutability of input parameters.
- A 'Lazy Copy On Write' scheme is used for cloning arrays.
- The underlying array storage consists of one-dimensional arrays, which can be accessed via simple pointer arithmetic. Even C# allows one to circumvent mandatory array bounds checks and to implement efficient loop unrolling this way. In the end, tight loops over large arrays get JIT compiled to native code that is very competitive with similar code written in C/C++.
- Built-in functions in ILNumerics efficiently spread the workload to all available cores. A hand-tuned thread pool implementation helps minimizing threading overhead.
- In many cases, ILNumerics knows from the context of a function when an operation can work in place, hence saving the allocation of temporary storage.
- Last but not least, the .NET framework has standard interfaces for interacting with external libraries. The Intel MKL® is used by default for all LAPACK related linear algebra routines, FFT and the like.

2.2. Mimicking the C# `using` statement

C# is a fine language, but our goal is to write PowerShell scripts. ILNumerics' heavy reliance on scopes poses a potential problem. The PowerShell language has a notion of scopes, but there's no direct counterpart to a `using` statement.

A typical C# method implemented using ILNumerics looks like this:

```
1 public static ILRetArray<double> FreqPeaks(
      ILInArray<double> in Data, ILInArray<double> inWindow, double sampFreq = 44100,
      ILOutArray<double> frequencies = null )
```

```
  {
5     using (ILScope.Enter(inData, inWindow)) {

          ILArray<double> retLength = min(ceil(inData.Length / 2.0 + 1), 5.0);
          ILArray<double> magnitudes = abs(fft(inData * inWindow));
          magnitudes = magnitudes[r(0,end / 2 + 1)];
10
          ILArray<double> indices = empty();
          ILArray<double> sorted = sort(magnitudes, indices, descending:true);
          if (!isnull(frequencies))
              frequencies.a =
15                (sampFreq / 2.0 / magnitudes.Length * indices)[r(0,retLength-1];
          return magnitudes[r(0,retLength-1)];
      }
  }
```

The C# compiler will transform this into:

```
 1 public static ILRetArray<double> FreqPeaks(
      ILInArray<double> inData, ILInArray<double> inWindow, double sampFreq = 44100,
      ILOutArray<double> frequencies = null )
  {
5     IDisposable scope = null;

      try
      {
          scope = ILScope.Enter(inData, inWindow);
10        ILArray<double> retLength = min(ceil(inData.Length / 2.0 + 1), 5.0);
          ILArray<double> magnitudes = abs(fft(inData * inWindow));
          ...
      }
      finally
15    {
            // this will dispose inData, inWindow and
            // all local arrays created within 'try {..}'
            if (scope != null) {
                scope.Dispose();
20          }
      }
  }
```

The `using` statement on line 5 in the first listing is translated into the call to `Dispose()` in the `finally` block shown in the second listing.

Of course, we can do all the .NET stuff in PowerShell, but there's another syntactic twist, or shall we say *serendipity*, which helps to make this look and feel right in PowerShell. Formally, the `using` statement has the appearance of a function. The arguments are `ILScope.Enter(inData, inWindow)` and a PowerShell scriptblock `{ ... }`. (Remember the syntax for invoking a PowerShell function `fun` with two arguments `a` and `b`? It's `fun a b` , not `fun(a,b)` or `fun a,b` .)

In September 2008, Keith Dahlby posted an example ([Dahlby2008]) of a PowerShell `using` function to his blog. His code is shown in Appendix B, *A `using` Statement for PowerShell*. Fortunately, we didn't have to come up with that on our own. Thanks Keith!

2.3. Making it look pretty

With the `New-Object` cmdlet, creating .NET objects in PowerShell is pretty straightforward. However, working with .NET methods, especially static methods, can be a pretty verbose affair. It might be good memory training, but the third time you type

`[System.IO.File]::ReadAllBytes(...)`, it'll be hard to resist the thought, "There must be a better way!"

Currently, the PSH5X module returns HDF5 dataset and attribute values as .NET arrays. To use them with ILNumerics, we must cast them as `ILArray`s. Just picture yourself typing `$a = [ILNumerics.ILArray[single]] Get-H5DatasetValue ...` more than twice. Not good.

PowerShell has a facility called *type accelerators* (see [Payette2011]) that let's you say `[psobject]` instead of `[System.Management.Automation.PSObject]`. Unfortunately, PowerShell doesn't offer a direct out-of-the-box capability to define our own accelerators. Fortunately, there is no shortage of sharp minds in the PowerShell community; in April 2012, Kirk Munro published his article *Creating and using type accelerators and namespaces the easy way* in the PowerShell Magazine [Munro2012], which gave us the final piece to pull this off. Kirk even wrote a nice PowerShell module, `TypeAccelerator`, which you can download from Codeplex [PSTX]. We use his module to define the ILNumerics accelerators shown in Appendix C, *`ILAccelerators.ps1`*. See Appendix E, *K-Means in PowerShell*, for a full-blown example which shows the accelerators in action.

3. Finale

With all the pieces in place it's all down to a few lines of PowerShell. In the example below, we open an HDF-EOS5 file, `HIRDLS-Aura_L2_v02-04-09-c03_2008d001.he5`, with data from the HIRDLS instrument [HIRDLS] and select a swath (see [WynFort1995]) of temperatures at

```
/HDFEOS/SWATHS/HIRDLS/Data Fields/Temperature
```

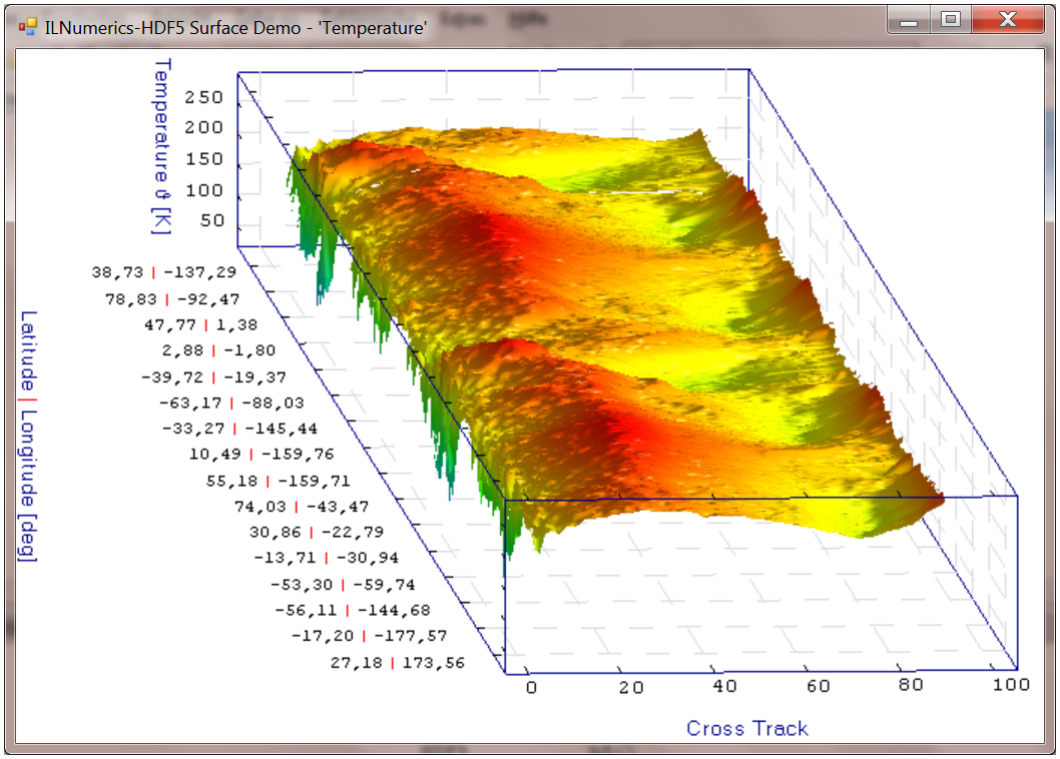
The dataset's fill value is stored in the `_FillValue` attribute of the dataset and we use `ILArray<T>::SetRange()` to replace them with `NaN`. The latitude and longitude labels are populated from the datasets at

```
/HDFEOS/SWATHS/HIRDLS/Geolocation Fields/Latitude
/HDFEOS/SWATHS/HIRDLS/Geolocation Fields/Longitude
```

Below is the full script and see Figure 1, “A Swath with HIRDLS temperature data.”, for the result.

```
Import-Module HDF5
Import-Module ILNumerics

$ScriptDir = Split-Path $MyInvocation.MyCommand.Path
# include drawing helper (see Appendix D)
. $ScriptDir\Surface.ps1

# create a new drive 'h5' backed by an HDF5 file and read the dataset value
New-H5Drive h5 "$ScriptDir\HIRDLS-Aura_L2_v02-04-09-c03_2008d001.he5"
[ilf32] $data1 = Get-H5Value 'h5:\HDFEOS\SWATHS\HIRDLS\Data Fields\Temperature'
[ilf32] $lab1 = Get-H5Value 'h5:\HDFEOS\SWATHS\HIRDLS\Geolocation Fields\Latitude'
[ilf32] $lab2 = Get-H5Value 'h5:\HDFEOS\SWATHS\HIRDLS\Geolocation Fields\Longitude'

# mark fill values as NaN values
$fill = Get-H5Value 'h5:\HDFEOS\SWATHS\HIRDLS\Data Fields\Temperature' '_FillValue'
$data1.SetRange([Single]::NaN, [ilmath]::eq($data1, $fill))
$data2.SetRange([Single]::NaN, [ilmath]::eq($data2, $fill))
$lab1.SetRange([Single]::NaN, [ilmath]::eq($lab1, $fill))
$lab2.SetRange([Single]::NaN, [ilmath]::eq($lab2, $fill))

Remove-H5Drive h5

# plot the first 750 columns of the data
$panel = Surface $data1['0:100;0:750'] $lab1['0:750'].Concat($lab2['0:750'],1)
```

Figure 1. A Swath with HIRDLS temperature data.



4. Summary

When we started this project, we were confident that with PowerShell, PSH5X, and ILNumerics we had all the necessary pieces to read data from an HDF5 file, to transform the data efficiently if needed, and to render them. What we didn't know was how well the pieces would fit together. It was clear that PowerShell was technically capable to serve as the glue, but we were unsure if the result would be some unsightly mess that we'd rather not talk about. We are glad to report that we not only achieved our goal, but that the solution also meets minimal aesthetic and health standards.

The take-home message is this: PowerShell might be seen as the Cadillac of administrators, but we don't need to ask for the keys to take it out for a spin. Hotwired with the HDF5 module and ILNumerics you are in for a comfortable ride to any corner of your data on the Windows platform.

A. ILNumerics Rules

Figure A.1. General rules for ILNumerics

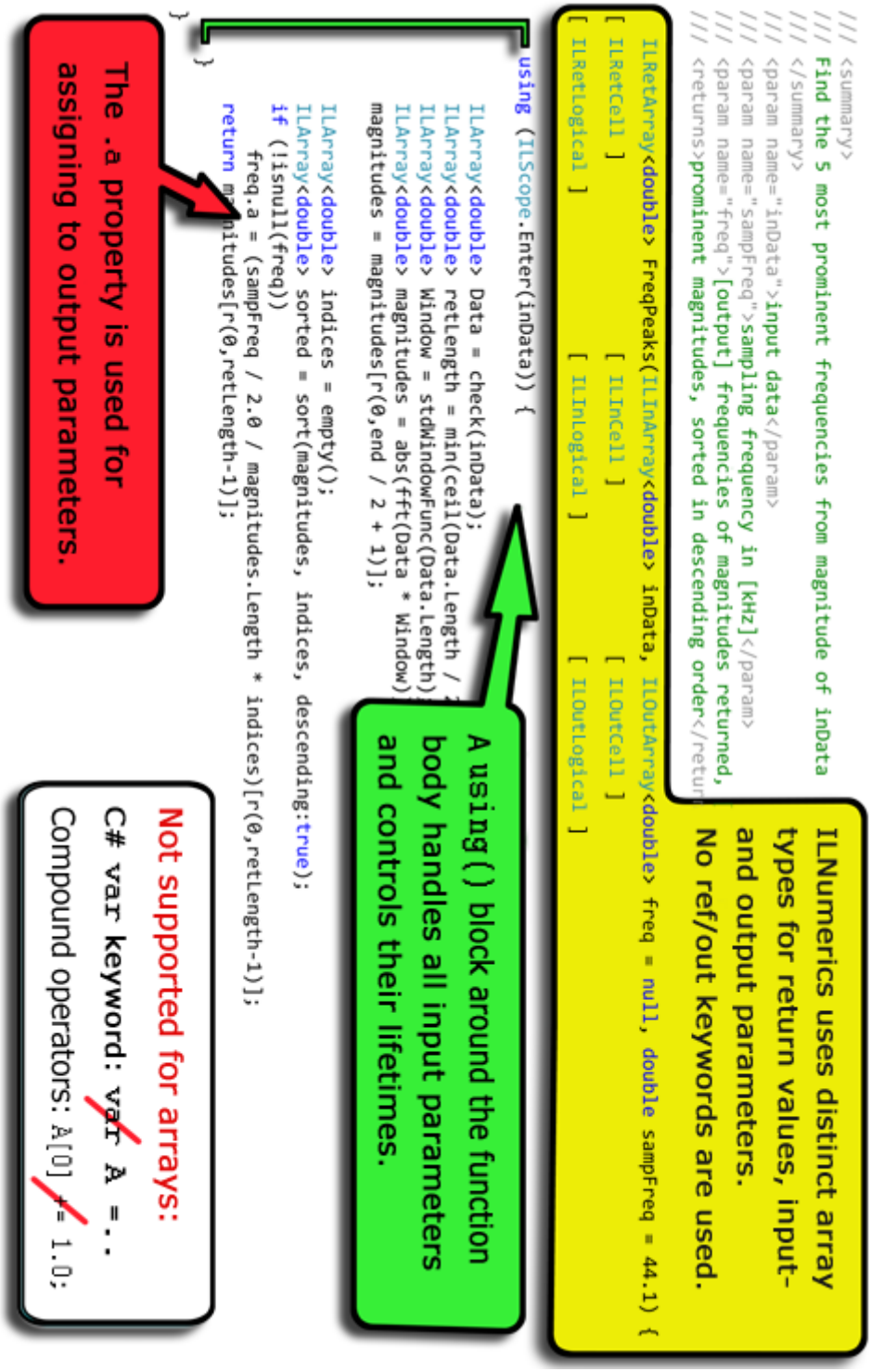


Figure A.1, “General rules for ILNumerics” gives an overview of the coding rules required to get reliable performance - even for demanding computational routines. By following those simple rules, ILNumerics will keep track of all array storage, provide value semantics for function parameters, clean up temporaries deterministically and much more [http://ilnumerics.net/$GeneralRules.html].

B. A `using` Statement for PowerShell

```
#
# This is Keith Dalby's script from
# http://solutionizing.net/2008/09/21/multi-purpose-powershell-using-function/
#

Function ILUsing {
    param (
        $inputObject = $(throw "The parameter -inputObject is required."),
        [ScriptBlock] $scriptBlock
    )

    if ($inputObject -is [string]) {
        if (Test-Path $inputObject) {
            [System.Reflection.Assembly]::LoadFrom($inputObject)
        } elseif($null -ne (
              New-Object System.Reflection.AssemblyName($inputObject)
              ).GetPublicKeyToken()) {
            [System.Reflection.Assembly]::Load($inputObject)
        } else {
            [System.Reflection.Assembly]::LoadWithPartialName($inputObject)
        }
    } elseif ($inputObject -is [System.IDisposable] -and $scriptBlock -ne $null) {
        Try {
            &$scriptBlock
        } Finally {
            if ($inputObject -ne $null) {
                $inputObject.Dispose()
            }
            Get-Variable -Scope script |
                Where-Object {
                    [object]::ReferenceEquals($_.Value.PSBase, $inputObject.PSBase)
                } |
                Foreach-Object {
                    Remove-Variable $_.Name -scope script
                }
        }
    } else {
        $inputObject
    }
}
```

C. `ILAccelerators.ps1`

```
Import-Module TypeAccelerator

Add-TypeAccelerator ilf32 ILNumerics.ILArray[single]
Add-TypeAccelerator ilif32 ILNumerics.ILInArray[single]
Add-TypeAccelerator ilof32 ILNumerics.ILOutArray[single]
Add-TypeAccelerator ilrf32 ILNumerics.ILRetArray[single]

Add-TypeAccelerator ilf64 ILNumerics.ILArray[double]
Add-TypeAccelerator ilif64 ILNumerics.ILInArray[double]
Add-TypeAccelerator ilof64 ILNumerics.ILOutArray[double]
Add-TypeAccelerator ilrf64 ILNumerics.ILRetArray[double]

Add-TypeAccelerator ili8 ILNumerics.ILArray[SByte]
Add-TypeAccelerator ili16 ILNumerics.ILArray[Int16]
```

```
Add-TypeAccelerator ili32 ILNumerics.ILArray[Int32]
Add-TypeAccelerator ili64 ILNumerics.ILArray[Int64]

Add-TypeAccelerator ilu8 ILNumerics.ILArray[Byte]
Add-TypeAccelerator ilu16 ILNumerics.ILArray[UInt16]
Add-TypeAccelerator ilu32 ILNumerics.ILArray[UInt32]
Add-TypeAccelerator ilu64 ILNumerics.ILArray[UInt64]

Add-TypeAccelerator ilmath ILNumerics.ILMath
Add-TypeAccelerator ilsettings ILNumerics.Settings
Add-TypeAccelerator ilscope ILNumerics.ILScope
```

D. Surface.ps1

```
Function Surface {
      param(
          # Height Data
          [ilif32] $data,
          # label values
          [ilif32] $labels)
          ILUsing ([ilscope]::Enter($data,$labels)) {

              [void][reflection.assembly]::LoadWithPartialName("System.Windows.Forms")
              [System.Windows.Forms.Application]::EnableVisualStyles();

              $form = New-Object Windows.Forms.Form
              $form.Text = 'ILNumerics-HDF5 Surface Demo - ''Temperature'''
              $form.Size = New-Object System.Drawing.Size(700,500)

              $ilpanel = [ILNumerics.Drawing.ILPanel]::Create()
              $form.Controls.Add($ilpanel)
              $form.Add_Shown({$form.Activate()})
              $surf = New-Object `
                  ILNumerics.Drawing.Plots.ILLitSurface($ilpanel, $data)
              $ilpanel.Graphs.AddPlot($surf)

              # some configuration - default view: from some angle to middle
              # of surface
              [Int32] $midX = $data.S[1] / 2
              [Int32] $midY = $data.S[0] / 2
              [float] $minZ = 0.0
              [float] $maxZ = 0.0
              $data.GetLimits([ref]$minZ, [ref] $maxZ)
              [Int32] $midZ = [Int32](($maxZ - $minZ) / 2)
              $ilpanel.DefaultView.SetDeg(86,50,1000)
              $ilpanel.DefaultView.LookAt = New-Object `
                  ILNumerics.Drawing.ILPoint3Df($midX, $midY, $midZ)

              $ilpanel.AspectRatio = `
                  [ILNumerics.Drawing.AspectRatioMode]::StretchToFill
              $ilpanel.Axes.XAxis.Label.Text = `
                  'Latitude \color{red}|\reset Longitude [deg]'
              $ilpanel.Axes.YAxis.Label.Text = 'Cross Track '
              $ilpanel.Axes.ZAxis.Label.Text = 'Temperature \vartheta [K]'
              $ilpanel.Lights[0].Position = New-Object `
                  ILNumerics.Drawing.ILPoint3Df(2000,1500,1500)
              $surf.Colormap = New-Object `
                  ILNumerics.Drawing.Misc.ILColormap ( `
                      [ILNumerics.Drawing.Colormaps]::ILNumerics  )
              $ilpanel.PlotBoxScreenSizeMode = `
                  [ILNumerics.Drawing.PlotBoxScreenSizeMode]::StrictOptimal
              # create custom labels
```

```
            $xaxisticks = `
               $ilpanel.Axes[[ILNumerics.Drawing.AxisNames]::XAxis].LabeledTicks
            $xaxisticks.Mode = [ILNumerics.Drawing.TickMode]::Manual
            $xaxisticks.Clear()
            $pickticks = (0..18 | ForEach-Object { $_ * 50})
            foreach ($i in $pickticks) {
                $xaxisticks.Add( $i,"\fontsize{-2}" `
                    + ("{0:f2} \color{{red}}|\color{{black}} {1:f2}" `
                        -f [float]$labels["$i;0"], [float]$labels["$i;1"]))
            }

            $form.ShowDialog()
        }
    }
```

E. K-Means in PowerShell

In the following listing, a sample PowerShell implementation of the popular k-means algorithm is given.

```
Import-Module ILNumerics

################################################################################
# <summary>
# k-means clustering: find clusters in data matrix X
# </summary>
# <param name="X">Data matrix, data points are expected as columns</param>
# <param name="k">Number of clusters</param>
# <returns>Vector of length n with indices of the clusters which were assigned
#    to each datapoint</returns>
################################################################################

Function KMeans
{
    param([ilif64] $X, [int] $k)

    ILUsing ([ilscope]::Enter($X, $k)) {

        if([ilmath]::isnull($X) -or -not $X.IsMatrix) {
            Write-Error 'X must be a matrix'
            exit
        }
        if ($X.IsEmpty) {
            return [ilrf64] [ilmath]::empty([ilmath]::size(0, $n))
        }
        if ($k -lt 1) {
            Write-Error 'number of clusters k must be positive'
            exit
        }
        if ($X.S[1] -lt $k) {
            Write-Error "too few datapoints provided for $k clusters"
            exit
        }
        $d = $X.S[0]; $n = $X.S[1]

        # initialize centers by using random datapoints
        [ilf64] $pickIndices = [ilmath]::empty();
        [ilmath]::sort([ilmath]::rand(1,$n), $pickIndices, 1, $False).Dispose()
        [ilf64] $centers.a = $X.Subarray((':', $pickIndices["0:$($k-1)"]))

```

```
            [ilf64] $classes = [ilmath]::zeros(1, $n)
            [ilf64] $oldCenters = $centers.C

            $maxIterations = 10000
45          while ($maxIterations-- -gt 0)
            {
                # assign every datapoint to nearest center
                for ($i = 0; $i -lt $n; ++$i)
                {
50                  # nested 'using' block for efficient memory performance
                    ILUsing ([ilscope]::Enter()) {

                        [ilf64] $minDistIdx = [ilmath]::empty()
                        # find the 'min'imal cluster distance
55                      [ilmath]::min([ilmath]::distL1($centers, $X[":,$i"]), `
                            # take the cluster index ... `
                            $minDistIdx, 1).Dispose()
                        # and use it as i-th class assignment
                        $classes[$i] = [double]$minDistIdx[0]
60                  }
                }
                # update the centers
                for ($i = 0; $i -lt $k; ++$i)
                {
65                  ILUsing ([ilscope]::Enter()) {
                        # find indices of all matching class members
                        $r = [ilmath]::find([ilmath]::eq($classes, [double]$i))
                        # extract corresponding member array
                        $inClass = [ilf64] $X.Subarray((':', $r))
70
                        if ($inClass.IsEmpty) {
                            $centers[":,$i"] = [double]::NaN
                        } else {
                            # compute the new cluster center (mean)
75                          $centers[":,$i"] = [ilf64] [ilmath]::mean($inClass, 1)
                        }
                    }
                }
                # if cluster stayed the same - break..
80              if ([ilmath]::allall([ilmath]::eq($oldCenters, $centers))) {
                    break;
                }
                # keep current centers for next round
                $oldCenters.a = $centers.C
85          }
            # must return class assignments as proper array type!
            # i.e. 'ilrf64' as alias for ILRetArray<double>
            , [ilrf64] $classes
        }
90  }
```

The algorithm takes as input a set of datapoints and the expected number of clusters k. After parameter checking and initialization, the main loop alternates between two steps:

1. Assign each datapoint to the cluster with the nearest cluster center.

2. Recompute the cluster centers.

The algorithm terminates when the centers seize to shift or a given number of iterations (e.g., 10,000) has been reached.

We used a C# version as the template for the PowerShell implementation. The differences are minor indeed. Powershell as a type-promiscuous language [Payette2011] saves us a bit of typing. (No pun

intended.) On the other hand, ILNumerics relies heavily on explicit array types. This is where the PowerShell accelerators come in handy. PowerShell's `using` block subsitute can be nested just like its C# cousin and produces the same effect; it limits the scope and minimizes memory pool size for long running loops. One exception to the otherwise harmonious transition from C# is that PowerShell doesn't leave much room for "syntactic sugar". Operators such as `+`, `-`, `||`, `==`, etc. must be replaced with static functions and, in the absence of classes, there are no derived classes.

Bibliography

[Dahlby2008] Keith Dahlby. *Multi-Purpose PowerShell Using Function [http://goo.gl/GXH1f]*. September 21, 2008.

[Finke2012] Douglas Finke. *Windows PowerShell for Developers [http://goo.gl/nrUIH]*. Enhance your productivity and enable rapid application development. O'Reilly. 2012.

[h5py] *A Python interface to the HDF5 library [http://code.google.com/p/h5py/]*. Andrew Collette. 2012.

[HDF5] *The Home of HDF5 [http://www.hdfgroup.org/HDF5/]*. The HDF Group. 2012.

[HDF-EOS] *HDF-EOS Tools and Information Center [http://hdfeos.org/]*. NASA/The HDF Group. 2012.

[HIRDLS] *HIgh Resolution Dynamics Limb Sounder [http://www.eos.ucar.edu/hirdls/]*. UCAR. 2008.

[HennPatt2012] *Computer Architecture - A Quantitative Approach* . John L. Hennessy and David A. Patterson. Morgan Kaufmann. 2012.

[HPCWith.NET] *High Performance Computing With .NET [http://goo.gl/lplWd]* . Haymo Kutschbach. 2012.

[ILNumerics] *Numeric Computing for Applications [http://ilnumerics.net/]*. Haymo Kutschbach. 2012.

[Munro2012] *A day in the life of a Poshoholic: Creating and using type accelerators and namespaces the easy way [http://goo.gl/KzK6k]*. Kirk Munro. 27 April, 2012.

[Numpy] *Scientific Computing Tools for Python [http://numpy.scipy.org/]*. Numpy developers. 2012.

[Payette2011] Bruce Payette. *Windows PowerShell in Action, Second Edition [http://www.manning.com/payette2/]*. Manning. May 18, 2011.

[PSTX] *PowerShell Type Accelerators [http://pstx.codeplex.com/]*. poshoholic. 2012.

[PSH5X] *HDF PSH5X Project [http://www.hdfgroup.org/projects/PSH5X/]*. The HDF Group. 2012.

[RIIA] *Resource Acquisition Is Initialization [http://goo.gl/OfxhE]*. Wikipedia. 2012.

[using Statement] *`using` Statement (C# Reference) [http://goo.gl/llPgm]*. MSDN. 2012.

[VPIC] *Sifting Through a Trillion Electrons [http://goo.gl/JuL2w]*. Berkeley researchers design strategies for extracting interesting data from massive scientific datasets. Lawrence Berkeley National Laboratory. 29 June 2012.

[WMF3] *Windows Management Framework 3.0 [http://goo.gl/rhdAJ]*. Microsoft. 4 September 2012.

[WynFort1995] *The HDF-EOS Swath Concept [http://goo.gl/h4Otf]*. White Paper 170-WP-003-001. Hughes Applied Information Systems. December 1995.