Image Registration with Automatic Computation of Gradients

Release 1.0

E. G. Kahn[1] and L. H. Staib[2]

July 29, 2008

[1]The Johns Hopkins University Applied Physics Laboratory, Laurel, Maryland
[2]Yale University, New Haven, Connecticut

Abstract

Many image registration algorithms are formulated as optimization problems with a gradient descent based solver, One difficulty with designing and implementing such methods is in the implementation of the gradient computation. This process can be time-consuming and error-prone. In addition some functions do not have gradients that can be expressed in symbolic form. Automatic differentiation is useful for computing gradients of complicated objective functions. It moves the burden of computing gradients from the programmer to the computer. So far, AD has not been exploited for use in image registration. This paper describes a software library the authors have developed to automate the process of computing gradients of registration objective functions. This can alleviate the job of registration designers somewhat and potentially make it easier to design better registration algorithms.

Contents

1 Introduction

In ITK's current registration framework, in order to introduce a new metric it is often desired to compute gradients and therefore one needs to override the function called `GetValueAndDerivative`. Additionally in order to introduce new transforms, one needs to override `GetJacobian`. Unfortunately, coding such derivatives is time consuming, error prone, and sometimes impossible when the objective function cannot

be expressed in symbolic form. The option of using finite differences is too slow when dealing with large numbers of parameters. It would be nice if we could avoid having to write these functions and yet still compute an exact gradient within a reasonable amount of time. Fortunately, there is a solution known as Automatic Differentiation (AD) [3]. Automatic differentiation is a method for overcoming such problems by having the computer automatically compute gradients in a fast and efficient way. In addition, because automatic differentiation alone does not scale well to large problems due to large memory requirements, another scheme known as "checkpointing" can be combined with AD to allow the gradient evaluation of functions of virtually any size[2, 5]. To date, the use of AD with checkpointing does not appear to have become popular in image registration problems. In this paper, we describe the integration of automatic differentiation with ITK. This paper assumes the reader has a good understanding of AD as a full discussion of these issues is beyond the scope if this work. For more details, see References [3, 4, 5, 7].

This paper is organized as follows. In Section 2, we describe how we integrated AD with ITK's registration framework. In Section 3 we describe some experiments conducted with our method and Section 4 provides a conclusion.

2 ITK Implementation

There are several open source projects available that implement automatic differentiation. The package we used is ADOLC by A. Griewank [4] which works by operator overloading in C++. Additionally the same author has a separate package known as Revolve [5] which implements the checkpointing algorithm. Unfortunately, these two packages had not previously been merged together (to our knowledge). Therefore it was necessary to combine them together before using them in ITK [6]. The result is a library called adcheck located in the directory src/adcheck. This library is completely independent of ITK and can be used to compute gradients of any scalar objective function.

In order to integrate ADOLC and Revolve, we had to modify some of the code from ADOLC, but almost no changes were made to the `revolve` function. (For the complete list of changes made to ADOLC see the file CHANGES in the src/adcheck directory.) In addition, we wrote two new C++ classes to serve as an API for the library.[1] The two new classes we wrote are called `ObjectiveFunctionBase` and `Revolve` (not to be confused with the C function `revolve`), which are both in a namespace called `adolc`. `adolc::ObjectiveFunctionBase` is an abstract class and the coder must create a derived class from this and define several methods. The most important of these is `Function` which implements the actual objective function. The second class is `adolc::Revolve` which is a driver class that runs the checkpointing routine. (This class is basically a C++ rewrite of the file `example.c` which is distributed together with the `revolve` package). This class contains a pointer to `adolc::ObjectiveFunctionBase`. An instance of `adolc::Revolve` and of the class derived from `adolc::ObjectiveFunctionBase` must be created. A pointer to the derived `adolc::ObjectiveFunctionBase` is then passed to `adolc::Revolve`. Various parameters can also be set in the `adolc::Revolve` class such as the number of checkpoints. Then `adolc::Revolve`'s method `Evaluate` is called to compute the function value and/or gradient.

(Note: To clarify, there are two objects which are called "revolve": (1) the C *function* `revolve` by Griewank (with lowercase "r"), and (2) our new C++ *class* `Revolve` (with uppercase "R") in the namespace `adolc`, which is essentially a C++ interface to the C function `revolve`.)

Once we created our combined automatic differentiation and revolve library, we then integrated it

[1] Another software library combining ADOLC with checkpointing, from which some aspects of our implementation were inspired, was written by Andrew Mauer-Oats of Northwestern University [8]. However, it is not integrated with the `revolve` function so we did not use it.

with ITK's registration framework by rewriting the metrics and transforms to make use of the adcheck library. As mentioned above, to use the adcheck library, the coder must create a derived class from the abstract `adolc::ObjectiveFunctionBase`. However, ITK works with a similar model: all metrics are derived from `itk::ImageToImageMetric` which is in turn derived from `itk::SingleValuedCostFunction`. Fortunately, C++ supports multiple inheritance and we derive all metrics from both classes. We created a special base class for this purpose called `itkek::ADRevolveImageToImageMetricBase` which directly descends from these two classes. All our rewritten metrics descend from `itkek::ADRevolveImageToImageMetricBase`. See Figure 1 for the class diagram. For the transforms, we adopted a similar strategy and created a new `itkek::ADRevolveTransformBase` whose parent class is `itk:TransformBase` and all the rewritten transforms were derived from this class. See Figure 2.

All our rewritten transform classes can be found in the directory src/Common and our rewritten metric classes can be found in the directory src/Algorithms. These classes are placed in the namespace `itkek`. We rewrote the mean squares and mutual information metrics to make use of AD:

- `itk::MeanSquaresImageToImageMetric -> itkek::ADRevolveMeanSquaresImageToImageMetric`
- `itk::MattesMutualInformationImageToImageMetric -> itkek::ADRevolveMattesMutualInformationImageToImageMetric`

In addition, we rewrote the following transforms, including rigid, affine, b-spline and radial basis functions:

- `itk::Euler3DTransform -> itkek::ADRevolveEuler3DTransform`
- `itk::MatrixOffsetTransformBase -> itkek::ADRevolveMatrixOffsetTransformBase`
- `itk::BSplineDeformableTransform -> itkek::ADRevolveBSplineDeformableTransform`
- `itk::KernelTransform -> itkek::ADRevolveRBFTransform`

Our new version of `itk::KernelTransform` is for the special case of a thin plate spline radial basis function transform in 3D which uses a GMRES [1] linear system iteration for solving for the coefficients. We called this new class `ADRevolveRBFTransform`. Note that it not yet possible in ITK to compute gradients of metrics that use a `KernelTransform` without resorting to finite differences since its `GetJacobian` function is not yet implemented, as of the most recent version of ITK (3.6.0). The approach of this paper provides an alternate way of computing gradients of such metrics.

3 Experiments

To test this software, we provide a test script in the accompanying software which includes at least one test for each of the rewritten transforms and metrics. In this section we only show the results for the B-Spline and thin plate spline transforms using the sum of squared difference metric to register a "planet" to a sphere (see Figure 3). Our planet is like a sphere except that it has seven "mountains" and seven "craters". Both images have dimensions $128 \times 128 \times 128$ with a spacing of 1. For the B-Spline transformation, a grid of 16x16x16 controls points was used, totaling 4096 control points. For the TPS transformation, 4000 control points were randomly distributed along the surface of the sphere. The resultant transformed images are

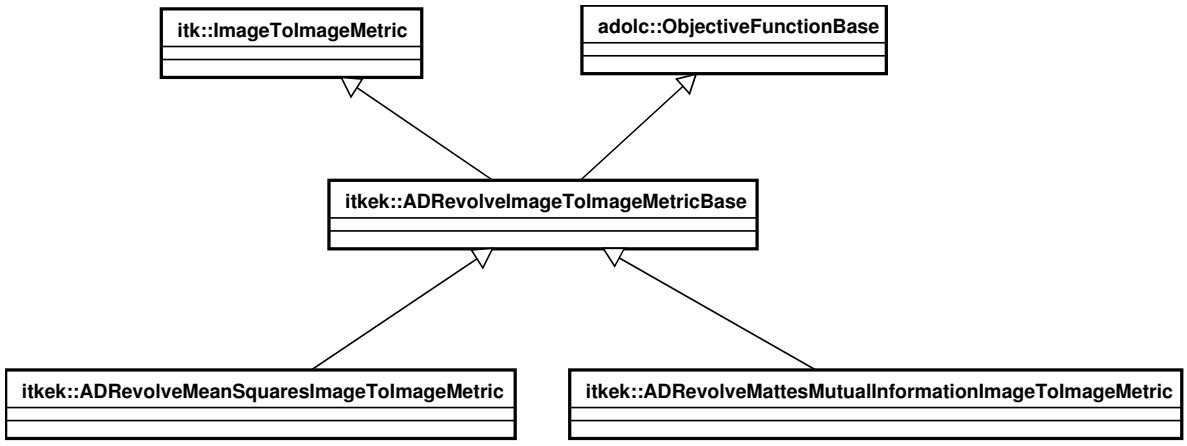

Figure 1: Class diagram of metrics.

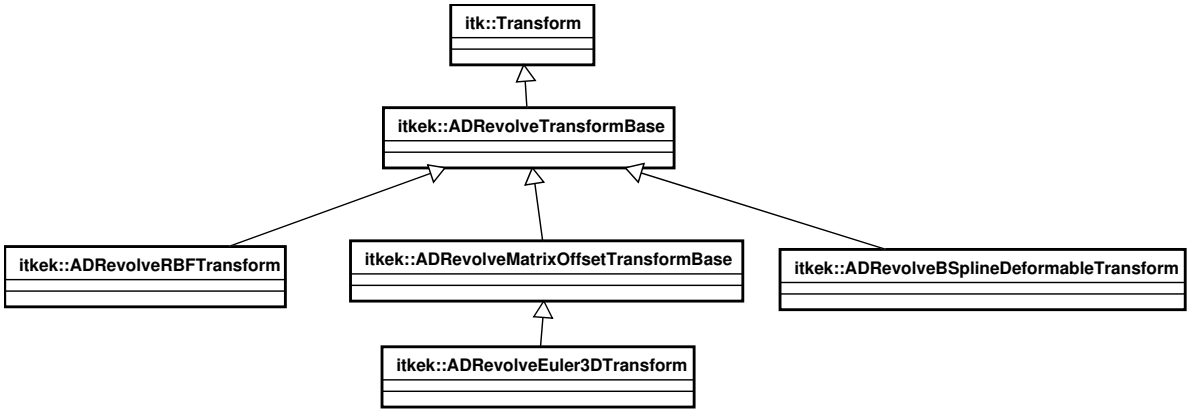

Figure 2: Class diagram of transforms.

shown in Figures 4 and 5. Note that B-Spline transformation is more local in nature and hence the warping only occurs in the vicinity of the mountains and craters. The TPS transform, on the other hand, is more global in nature and the entire space is warped. Note, the control points are only along the surface of the sphere and not spread equally out throughout the image.

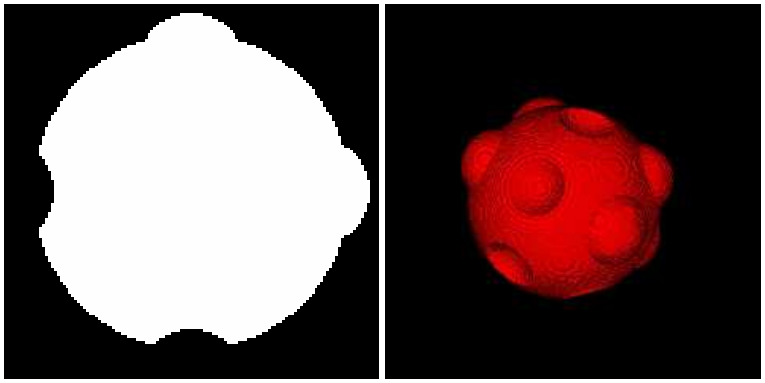

Figure 3: Slice and surface rendering of planet with “mountains” and “craters”.

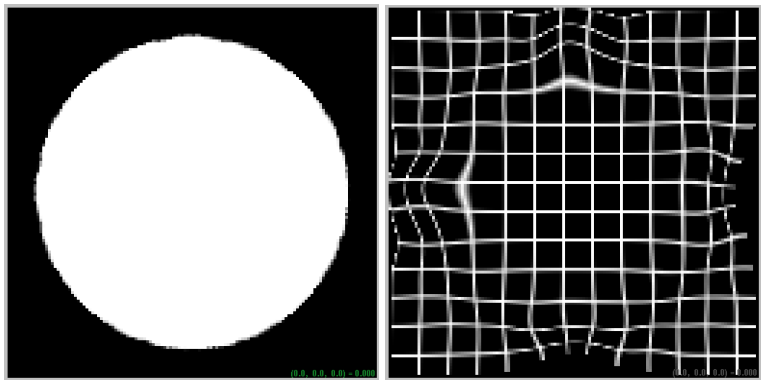

Figure 4: Slice of transformed image with corresponding warped grid image for B-spline grid of $16 \times 16 \times 16$ (4096 points), using a sum of squared metric.

4 Conclusion

We used automatic differentiation with checkpointing to implement a gradient descent based registration framework. In addition to reimplementing several metrics and transforms for which derivatives can be computed easily without recourse to automatic differentiation, we were also able to compute derivatives of metrics employing a kernel transform, such as thin plate splines, which is not yet possible in ITK unless one uses finite differences. We expect that the methodology of this paper can be used to more rapidly design new metrics or transforms without the need to write derivative functions.

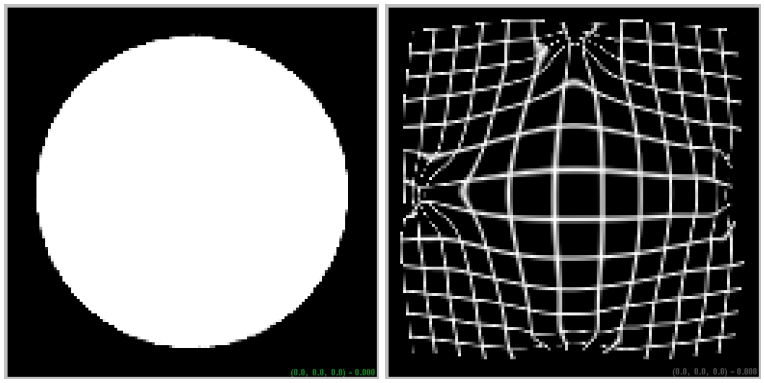

Figure 5: Slice of transformed image with corresponding warped grid image for TPS of 4000 points, using a sum of squared metric.

References

[1] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994. 2

[2] Andreas Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1:35–54, 1992. 1

[3] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, PA, 2000. 1

[4] Andreas Griewank, David Juedes, and Jean Utke. Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software*, 22(2):131–167, June 1996. 1, 2

[5] Andreas Griewank and Andrea Walther. Algorithm 799: Revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software*, 26(1):19–45, March 2000. 1, 2

[6] L. Ibanez and W. Schroeder. *The ITK Software Guide: The Insight Segmentation and Registration Toolkit*. Kitware, Inc., Albany, NY, http://www.itk.org, 2003. 2

[7] Eliezer Kahn. *Computational Strategies for Meshfree Nonrigid Registration*. PhD thesis, Yale University, December 2006. 1

[8] Andrew Mauer-Oats. Checkpoint, http://www.math.northwestern.edu/~amauer/projects/checkpoint/, 1997. 1