A Gradient Descent Trust Region Optimizer

Release 1.00

Rupert Brooks[1]

August 11, 2011

[1]Elekta Ltd, 2050 Bleury Street, Suite 200
Montreal, QC, Canada
rupert.brooks (at) elekta.com

Abstract

This document describes a new simple gradient descent optimizer which is a potential replacement or supplement for the `itk::RegularStepGradientDescentOptimizer`. The optimizer requires only the gradient of the parameters and uses a simple linear model internally. However, it follows the theoretical basis of a trust-region algorithm and is able to achieve greater efficiency on certain image registration cases.

Contents

ITK provides the `itk::RegularStepGradientDescentOptimizer` class for simple optimization problems, such as those occuring in low dimensional image registration. This optimizer has the advantage of extreme simplicity. However, in practice it can be somewhat difficult to tune correctly. In this paper, a trust-region gradient descent optimizer class is proposed. This algorithm is almost as simple, but is based on a more principled approach. This optimizer can be shown to be more efficient in terms of number of steps and more reliable in finding an optimum in somewhat difficult cases.

1 Trust-region Optimization

We are interested here in approaches for solving the unconstrained, multidimensional optimization problem

$$\phi_{opt} = argmin(f(\phi))$$

That is, we wish to find the set of parameters, ϕ, which minimizes the cost function, $f(\phi)$. We frame our discussion in terms of minimization, but maximization can of course be achieved by simply negating the cost function.

Trust-region optimization approaches solve this problem by creating a model of the cost function to be optimized near the current estimate of the solution. This local model is then solved to provide a new estimate of the solution. However, it is known that the model is only an accurate representation of the true objective function near the current estimate. Therefore, the next solution estimate is the best estimate generated from the model, constrained to be within the area where the model is considered valid, known as the trust-region.

The general trust-region algorithm is therefore as shown in Algorithm 1.

Algorithm 1 General Trust-Region Optimization algorithm [2]

```
 1: Set start position $\phi_0$; iteration counter $n = 0$; scaling matrix $S$; step size $s$
 2: repeat
 3:   Compute one or more of the cost function and its derivatives at the current position $\phi_{(n)}$
 4:   Generate a local model of the function, and a region on which the model is trusted.
 5:   Solve the trust-region subproblem to get the update to the parameters, $\Delta\phi_{(n)}$.
 6:   Compute the cost function at the solution of the trust-region subproblem.
 7:   if the solution is not sufficiently improved then
 8:     Reject the step, and shrink the trust-region.
 9:   else if the solution is better than expected then
10:     Accept the step, and enlarge the trust-region.
11:     Set $\phi_{(n+1)} = \phi_{(n)} + \Delta\phi_{(n)}$ and $n = n+1$
12:   else
13:     {the solution is acceptable relative to the model}
14:     Accept the step.
15:     Set $\phi_{(n+1)} = \phi_{(n)} + \Delta\phi_{(n)}$ and $n = n+1$
16:   end if
17: until the convergence criteria are reached
```

Trust region algorithms stand in contrast to line search methods, which work by choosing a descent direction - a direction in the parameter space which reduces the value of the objective function - and then minimizing the value of the objective function along that line using any of a number of one-dimensional search methods. Further theoretical discussion may be found in [2].

Trust-region methods are potentially useful for image registration problems because they tend to have fewer cost function evaluations, and image registration problems have cost functions that are notoriously computationally costly. In fact, we can see that the existing regular step gradient descent optimizer (Algorithm 2) can be viewed a form of trust region optimizer. The model is simply a plane, normal to the gradient, and the trust region is a circle with radius of the step size. The solution of the trust-region subproblem is always to step along the negative gradient direction up to the edge of that circle. The trust region is shrunk when the step fails to reduce the function, however, it can never be enlarged, nor does it ever truly reject a step.

Algorithm 2 Regular Step Gradient Descent Optimizer [3]

```
 1: Set start position $\phi_0$; iteration counter $n = 0$; scaling matrix $\boldsymbol{S}$; step size $s$
 2: repeat
 3:   Compute the gradient at the current position $\nabla D(\phi_{(n)})$
 4:   Compute the scaled gradient, $\nabla D^* = \boldsymbol{S} \times \nabla D(\phi_{(n)})$
 5:   if the scaled gradient has changed direction by more than 90 degrees then
 6:     Set $s = \frac{s}{2}$    {R}educe the step size
 7:   end if
 8:   Compute the update to the parameters, $\Delta\phi_{(n)} = \frac{\nabla D^*}{||\nabla D^*||} \cdot s$
 9:   Set $\phi_{(n+1)} = \phi_{(n)} + \Delta\phi_{(n)}$ and $n = n + 1$
10: until the convergence criteria are reached
```

It is proposed that following the formal trust region strategy will yield an optimizer similar in simplicity and ease of use, but with better performance. In this paper, we propose the `itk::GradientDescentTrustRegionOptimizer`, which uses the algorithm shown in Algorithm 3.

Initial values for the various constants which work well in most cases have been provided, and are summarized in Table 1. These values were chosen based on the recommendations in [2] and seem to work well for most applications.

Scaling is commonly applied to optimization algorithms in ITK to aid in convergence. Scaling can be viewed as a reparameterization of the optimation problem with a new set of parameters where $\phi_{new} = \boldsymbol{S}\phi_{old}$. In such a system, the gradient with respect to the new parameters will be $\boldsymbol{S}^{-1}\nabla_{\phi_{old}} D(\phi_{old})$ where D is the cost function. A gradient descent optimizer should take a unit step in this gradient direction, in the new parameters. Converting back to the old parameters requires an additional multiplication by $\boldsymbol{S}^{-1}$. It is important to note that in the `itk::RegularStepGradientDescentOptimizer`, both of these steps are combined into one and the gradient is simply divided by the scale factors. This is equivalent to a linear rescaling of the parameters by the square root of the scale factors. As the `itk::GradientDescentTrustRegionOptimizer` is intended to be a drop in replacement for the `itk::RegularStepGradientDescentOptimizer` this convention is kept. In the trust region case, however, the radius of the trust region must also be considered. The trust-region radius is computed in the scaled parameter space - thus it is scaled by the square root of the scaling factors. Further discussion of the role of the scale factors can be found in [1, Chap. 4].

This role of the trust region radius means that for the trust region optimizer, the scales have real meaning. That is, scaling an affine transformation by, for example, leaving the matrix parameter scales at 1, and reducing the translation parameter scales to a very small number is not equivalent to setting the matrix parameter scales to a large number and leaving the translation parameter scales at 1. In fact, this required a change to the scaling in the ITK example `MultiResImageRegistration2.cxx` in order to get it to work properly with both optimizers.

2 Testing and Results

The existing test for the scheme in ITK is appropriate for this optimizer, and has been used directly, with the addition of a test of the maximization mode. The optimizer can also be substi-

Algorithm 3 Gradient Descent Trust Region optimization.

```
**Set** start position $\phi_0$; iteration counter $n = 0$; initial trust-region radius $R$; scaling matrix $\boldsymbol{M}$
**Compute** $\nabla_{\phi} D(\phi_{(0)})$ and $D(\phi_{(0)})$
**repeat**
  **Compute** $\phi_{(n+1)}$ at intersection of negative scaled gradient and trust-region boundary.
  **Compute** Expected improvement $E_{(i)} = (\phi_{(n+1)} - \phi_{(n)}) \cdot \nabla_{\phi} D(\phi_{(n)}) - D(\phi_{(n)})$
  **Compute** $\nabla_{\phi} D(\phi_{(n+1)})$ and $D(\phi_{(n+1)})$
  **Compute** $\rho$, the ratio of real to expected improvement. $\rho = \frac{(D(\phi_{(n+1)}) - D(\phi_{(n)}))}{E_{(n)}}$
  **if** $\rho < c_0$ **then**
    {This is an unexpectedly poor result, so the model is wrong. Reject the step and shrink the trust region.}
    $R = \gamma_0 R$
  **else if** $\rho < c_1$ **then**
    {This is a poor result, but still a decrease. Accept the step and shrink the trust region.}
    $R = \gamma_1 R$
  **else if** $\rho < c_2$ **then**
    {The improvement is acceptable relative to the model, so accept the step}
    $n = n + 1$
  **else**
    {This improvement is good or excellent relative to the model. Accept the step, and expand the trust region}
    $n = n + 1; R = \min(\gamma_2 R, R_{max}$
  **end if**
**until** $n > N_{max}$ or $|\nabla_{\phi} D(\phi_{(n+1)})| < G_{min}$ or $R < R_{min}$
```

Name	Default	Range	Variable	Description
UpperDecreaseRatio	0.75	> 0	c_2	If the improvement ratio is better than this, increase the trust region size.
MiddleDecreaseRatio	0.1	$(0,1)$	c_1	If the improvement ratio is less than this, decrease the trust region size.
LowerDecreaseRatio	0.001	< 1	c_0	If the improvement ratio is less than this, reject the step, and decrease the trust region size.
RejectedStepDecreaseFactor	0.5	$(0,1)$	γ_0	Shrink the trust region by this factor when the step is very poor.
PoorStepDecreaseFactor	0.5	$(0,5)$	γ_1	Shrink the trust region by this factor when the step is poor.
StepIncreaseFactor	2.0	> 1	γ_1	Enlarge the trust region by this factor when the step is good.
MinimumStepLength	0.001	> 0	R_{min}	If the trust region shrinks smaller than this, stop the optimization.
InitialStepLength	1.0	> 0	R	The starting trust region radius.
MaximumStepLength	64.0	> 0	R_{max}	An upper limit on the trust region radius.
GradientMagnitudeTolerance	0.000001	> 0	G_{min}	If the gradient shrinks smaller than this, stop the optimization.
NumberOfIterations	100	> 0	N_{max}	Stop the optimization after this many iterations, even if not complete.

Table 1: Parameters of the Gradient Descent Trust Region Algorithm

	Regular Step		Trust Region	
Example	Iterations	Final Value	Iterations	Final Value
ImageRegistration1	18	0.00745293	23	1.24136e-06
ImageRegistration3	18	0.00745293	23	1.24136e-06
ImageRegistration4	50	-0.929332	23	-0.929464
ImageRegistration5	200	555.216 (FAIL)	17	43.8326
ImageRegistration6	23	43.8218	19	43.834
ImageRegistration7	76	53.5944	48	53.6376
ImageRegistration9	158	44.0226	115	43.8151
ImageRegistration12	23	68.6133	19	68.6355
ImageRegistration13	27	-1.34363	8	-1.34364
ImageRegistration18	19	86730.4	11	85974.2 (FAIL)
ImageRegistration20	57	43.6097	48	43.6489
MultiResImageRegistration1	74/83/87	-1.28378	20/15/9	-1.29405
MultiResImageRegistration2	47/46/60	-1.28631	12/12/17	-1.3007

Table 2: Results on Image Registration Examples from the ITK Book [3].

tuted for the gradient descent optimizer in several of the image registration examples[1][2]. Examples `ImageRegistration{1,3--7,9,12,13,18,20}` and `MultiResImageRegistration{1,2}` have been modified to select between the optimizers and are included with this submission for easy comparison. It is interesting to note that in each example, the `itk::GradientDescentTrustRegionOptimizer` resolves with similar accuracy as the `itk::RegularStepGradientDescentOptimizer`, but with fewer steps. This is summarized in the following table, and the actual command lines used are contained in an appendix.

3 Discussion

Several cases stand out in the examples as particularly interesting. ImageRegistration5 is a case where the regular step optimizer fails with the default arguments, mainly due to being unable to enlarge the step. ImageRegistration13 is a case where the regular step method accepts a poor step, and then has to work its way back to the solution. However, this policy of never moving to a less optimal solution is not always suitable. In both cases, the trust region approach clearly shows its advantages. In contrast, ImageRegistration18 uses the `itk::GradientDifferenceImageToImageMetric`, which has a noisy gradient. The trust region optimizer remains firmly stuck in a local optimum on this example, while the regular step approach is able to temporarily move to a poorer value allowing it to escape to find the global optimum.

This paper has described the `itk::GradientDescentTrustRegionOptimizer` which is intended as an improvement on the existing `itk::RegularStepGradientDescentOptimizer`. It is similar in intent, but follows a formal trust region optimization scheme. Probably the most significant advantage of this scheme, is that the algorithm can both increase and decrease its step size adaptively. On smooth functions, its tends to outperform the regular step method, however, it may be less suitable for noisy functions.

[1]ImageRegistration3 is nearly an identical example to ImageRegistration1.

[2]The intended arguments for ImageRegistration18 were not clear so a multimodal image pair was chosen from the example data arbitrarily.

References

[1] Rupert Brooks. *Efficient and Reliable Methods for Direct Parameterized Image Registration*. Ph.D. thesis, McGill University, 2008. 1

[2] Andrew R. Conn, Nicholas I. M. Gould, and Phillippe .L Toint. *Trust-Region Methods*. Society for Industrial and Applied Mathematics and Mathematical Programming Society, 2000. 1, 1

[3] Luis Ibanez, William Schroeder, Lydia Ng, and Josh Cates. *The ITK Software Guide: ITK V2.0*. Kitware Inc, Clifton Park, NY, USA, 2005. 2, 2

A Command lines used for testing

These code examples suppose that the variable `ITK_DATA` has been set with the path to the `<itkSource>/Examples/Data`, that the variable `ITK_BRAINWEB` has been set with the path to the ITK Brainweb data and that they are being executed in a unix style shell, in the build directory.

```
./ImageRegistration1 rs $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png
./ImageRegistration1 tr $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png

./ImageRegistration3 tr $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png
./ImageRegistration3 rs $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png

./ImageRegistration4 rs $ITK_DATA/BrainT1SliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png
./ImageRegistration4 tr $ITK_DATA/BrainT1SliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png

./ImageRegistration5 rs $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceR10X13Y17.png test.png
./ImageRegistration5 tr $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceR10X13Y17.png test.png

./ImageRegistration6 rs $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceR10X13Y17.png test.png
./ImageRegistration6 tr $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceR10X13Y17.png test.png

./ImageRegistration7 rs $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceR10X13Y17S12.png test.png
./ImageRegistration7 tr $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceR10X13Y17S12.png test.png

./ImageRegistration9 rs $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceR10X13Y17.png test.png
./ImageRegistration9 tr $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceR10X13Y17.png test.png
```

```
./ImageRegistration12 rs $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceR10X13Y17.png \
   $ITK_DATA/BrainProtonDensitySliceBorder20Mask.png test.png
./ImageRegistration12 tr $ITK_DATA/BrainProtonDensitySliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceR10X13Y17.png \
   $ITK_DATA/BrainProtonDensitySliceBorder20Mask.png test.png

./ImageRegistration13 rs $ITK_DATA/BrainProtonDensitySlice.png \
    $ITK_DATA/BrainProtonDensitySliceBorder20.png  test.png
./ImageRegistration13 tr $ITK_DATA/BrainProtonDensitySlice.png \
    $ITK_DATA/BrainProtonDensitySliceBorder20.png  test.png

./ImageRegistration18 rs $ITK_DATA/BrainT1SliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png
./ImageRegistration18 tr $ITK_DATA/BrainT1SliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png

./ImageRegistration20 rs $ITK_BRAINWEB/brainweb1e1a10f20.mha \
   $ITK_BRAINWEB/brainweb1e1a10f20Rot10Tx15.mha  test.mha
./ImageRegistration20 tr $ITK_BRAINWEB/brainweb1e1a10f20.mha \
   $ITK_BRAINWEB/brainweb1e1a10f20Rot10Tx15.mha  test.mha

./MultiResImageRegistration1 rs $ITK_DATA/BrainT1SliceBorder20.png
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png
./MultiResImageRegistration1 tr $ITK_DATA/BrainT1SliceBorder20.png
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png

./MultiResImageRegistration2 rs $ITK_DATA/BrainT1SliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png
./MultiResImageRegistration2 tr $ITK_DATA/BrainT1SliceBorder20.png \
   $ITK_DATA/BrainProtonDensitySliceShifted13x17y.png test.png
```