Hybrid methods

Since we have

- a method [steepest descent with line search] which guarantees global convergence but can stagnate near the minimum
- a method [Gauss-Newton] which is not guaranteed to converge, unless you're close to a minimum, but typically converges faster near the minimum

$\Longrightarrow$ we should try a hybrid method that combines the best features of both
like `fzero` for root-finding
We present as a trust region method [could also present as line search].

Trust region methods

The idea of trust region TR methods is to to locally minimize a model for the cost function, in a trust region of radius Δ about the current position, in which you think the model is good.
A pseudocode would be:

```
Initialize the trust region radius
Do until termination criteria satisfied
   solve the trust region subproblem
          [minimize the model inside the trust region]
   test the trial point x_t and the TR radius and
          decide to accept or not x_t or radius or both
   end
```

The trust region subproblem

At the current position, we want to minimize the quadratic model

$$\min m(\mathbf{x}) = C(\mathbf{x}_c) + \mathbf{s}^T \mathbf{J}(\mathbf{x}_c)^T \mathbf{R}(\mathbf{x}_c) + \frac{1}{2}\mathbf{s}^T \mathbf{J}(\mathbf{x}_c)^T \mathbf{J}(\mathbf{x}_c)\mathbf{s}$$

using the Gauss-Newton approximation to $\nabla^2 C$, subject to

$$||\mathbf{s}|| \leq \Delta$$

We either

- have the local minimizer inside the TR, in which case we don't see the TR constraint, and the minimizer is found from

  $$\mathbf{J}(\mathbf{x}_c)^T \mathbf{J}(\mathbf{x}_c)\mathbf{s} = -\mathbf{J}(\mathbf{x}_c)^T \mathbf{R}(\mathbf{x}_c)$$

  the Gauss-Newton step, or
- the minimizer falls outside the TR, so we must have the equality constraint

  $$||\mathbf{s}|| = \Delta$$

This second case we can solve using Lagrange multipliers [from MAST20009].

Lagrange multipliers

To minimize a function $m(\mathbf{x})$ subject to an equality constraint $c(\mathbf{x}) = 0$, we form the augmented cost function or Lagrangian

$$\mathcal{L}(\mathbf{x}, \lambda) = m(\mathbf{x}) - \lambda c(\mathbf{x})$$

Then the necessary conditions for a minimum are

$$\nabla_{\mathbf{x}} \mathcal{L} = \mathbf{0}$$

and the constraint equation.
In the present context, we have $c(\mathbf{x}) = \frac{1}{2}(\Delta^2 - \mathbf{s}^T\mathbf{s})$ and $m(\mathbf{x})$ is the quadratic model. Then

$$\nabla_{\mathbf{x}} \mathcal{L} = \mathbf{J}(\mathbf{x}_c)^T \mathbf{R}(\mathbf{x}_c) + \lambda \mathbf{s} + \mathbf{J}(\mathbf{x}_c)^T \mathbf{J}(\mathbf{x}_c)\mathbf{s} = \mathbf{0}$$

$$\left[\mathbf{J}(\mathbf{x}_c)^T \mathbf{J}(\mathbf{x}_c) + \lambda \mathrm{I}\right] \mathbf{s} = -\mathbf{J}(\mathbf{x}_c)^T \mathbf{R}(\mathbf{x}_c)$$

a weighted combination of Gauss-Newton (for small λ) and steepest descent (for large λ).

Levenberg-Marquardt method [1944/1963]

This weighted combination is called the Levenberg-Marquardt method and is a common method for nonlinear least squares problems. The trust region derivation is due to Moré [1977].

The idea is to adjust the value of λ so as to maintain a TR in which the quadratic model is good. In many implementations we never adjust Δ explicitly, only λ — the Levenberg-Marquardt parameter. We take a trial step; if it's good by some measure, we accept the step and perhaps alter λ; if not, we reject the step, alter λ and make a new trial step.

We must start with an initial value for λ — values of 10^{-2} or 10^{-3} are typical.

Choosing values of λ

Since $\mathbf{s}$ decreases as λ increases, if the step is successful we decrease λ [increase Δ] to make the step more like Gauss-Newton; if not, we increase λ [decrease Δ] to make the step more like steepest descent.

MATLAB defines as a successful step whether the cost function is reduced, then λ is divided by 10. If the cost function increases at the trial position, the step is rejected and λ is mutiplied by 10.

More sophisticated schemes are often used based on the actual reduction of the cost function, compared to the predicted reduction of the cost function i.e. how well the quadratic model is doing.

Properties

At each step, we don't solve the normal equations

$$\left[\mathbf{J}(\mathbf{x}_c)^T\mathbf{J}(\mathbf{x}_c) + \lambda \mathrm{I}\right]\mathbf{s} = -\mathbf{J}(\mathbf{x}_c)^T\mathbf{R}(\mathbf{x}_c)$$

Instead, we find the (linear) least squares solution of the overdetermined system

$$\begin{bmatrix} \mathbf{J}(\mathbf{x}_c) \\ \sqrt{\lambda \mathrm{I}} \end{bmatrix}\mathbf{s} = -\begin{bmatrix} \mathbf{R}(\mathbf{x}_c) \\ 0 \end{bmatrix}$$

Under mild assumptions, the Levenberg-Marquardt method

- converges globally to a local minimum
- converges quadratically in the final iterations

an efficient and robust method for nonlinear least squares

MATLAB by default uses a different trust region method, with Levenberg-Marquardt as an optional method.

Example

In MATLAB

the commands for solving NLSQ problems are in the Optimization App/Toolbox and are used in the Curve Fitting App/Toolbox.

They include:

- `lsqnonlin`
- `lsqcurvefit`

`lsqcurvefit` is mainly a more convenient wrapper to `lsqnonlin` for curve fitting applications. It calls `lsqnonlin` but requires different inputs from the user.

`lsqnonlin` requires the user to provide a function that returns the residual vector **R** at the current value of **x**, given the data $\{T_i, Y_i\}$.

`lsqcurvefit` requires the user to provide a function F that describes the nonlinear model

$$y \sim F(\mathbf{x}, \{T_i\})$$

at the current value of **x**, given the data $\{T_i\}$. The data $\{T_i, Y_i\}$ are also explicit input arguments.

End of Week 9