Eigen Bug Writeup

Issue: Assignment using `<<` and `=` do not yield the same results, when temporary expression objects are involved.

Minimal Reproducible Example

```
 1. #include <eigen3/Eigen/Dense>
 2. #include <iostream>
 3.
 4. int main() {
 5.  // system dimension
 6.  const unsigned int n = 9;
 7.  // random test vectors
 8.  Eigen::MatrixXd R = Eigen::MatrixXd::Random(n, n);
 9.  Eigen::VectorXd b = Eigen::VectorXd::Random(n);
10.  Eigen::VectorXd xa = Eigen::VectorXd::Zero(n),
11.                  xb = Eigen::VectorXd::Zero(n);
12.
13.
14.//--------------------------------------------------------------
15.  xa << R.triangularView<Eigen::Upper>().solve(b);
16.  xb = R.triangularView<Eigen::Upper>().solve(b);
17.
18.//--------------------------------------------------------------
19. std::cout << "LSE solution A: " << xa.transpose() << std::endl;
20. std::cout << "LSE solution B: " << xb.transpose() << std::endl;
21. std::cout << "Error compared between A and B: " << (xa - xb).norm()
22.            << std::endl;
23.}
```

On lines 10 and 11, `xa` and `xb` are zero-initialized to mitigate differences arising due to different memory contents.

Line 15 shows the initialization using `<<` and `,` which Eigen calls “comma initializers”.

Line 16 demonstrates the usual initialization using the `=` operator.

The two initialization methods lead to a small difference in the order of magnitude around 10^{-10} for a 9×9 matrix, which grows quickly with the matrix dimension. This explains why some students could not submit their solutions on code expert.

Analysis

Analysing the assembly produced by two different assignment methods, we find that assignment using `<<` leads to four function calls, while `=` only yields three thereof.

The first two calls are identical and are related to `TriangularView` and `solve`.

The remaining calls are different:
`=` simply finishes the evaluation of `solve` and, presumably (we have not found the corresponding assembly instructions yet), directly to the memory location of the variable.
`<<` 'fuses' `solve` and `CommaInitializer`, and then calls `CommaInitializer` again, presumably to store the value.

The difference between the two values arises from the fact that `<< + <expression>` is an expression on its own and not equivalent to a simple assignment.

This can easily be observed when we store the value of `cpp R.triangularView<Eigen::Upper>().solve(b)` to a variable and assign it using `<<`:

```
Eigen::VectorXd sol = R.triangularView<Eigen::Upper>().solve(b);
xa << sol;
xb = Eigen::VectorXd sol = R.triangularView<Eigen::Upper>().solve(b);
std::cout << (xa - xb).norm() << std::endl // == 0.0
```

Using this fact, we can formulate a strategy to mitigate this error.

Mitigation

Since the error arises in the evaluation of the comma initializer, the issue can be resolved by evaluating the calculation (in our example the call so `solve`) early:

```
xa << R.triangularView<Eigen::Upper>.solve(b).evaluate();
```

which, while not binary equivalent, does evaluate to the same expected value.