252-0538-00L, Spring 2018

Shape Modeling and Geometry Processing

Mass-Spring and FEM Simulation
Part 2

CRL

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Overview

- Overview of code for exercise 2
- Quick intro to AutoDiff, and how to use it in the assignment
- Questions for exercise 1

Questions

If you have questions about the assignment:

moritzge@inf.ethz.ch

Exercise 2

Neo-Hookean material model, strain energy density:

2D: $$\Psi_{\text{NH}} = \frac{\mu}{2}(tr(\mathbf{C}) - 2) - \mu \ln(J) + \frac{\lambda}{2}\ln(J)^2$$ (Assignment)

Compressible Neo-Hookean material:

3D: $$\Psi_{NH} = \frac{\mu}{2}(\text{tr}(\mathbf{C}) - 3) - \mu \ln J + \frac{\lambda}{2}\ln(J)^2$$ (Lecture)

Who is right??

Exercise 2.1

FEM simulation with gradient descent.

- How to get from energy *density* to Ψ_{NH} energy *W*?
  Answer is in lecture slides!

- Compute gradient with finite differences

- *Hint*. An issue which didn't occur using springs, is that of inverted elements. One way to deal with this issue, is to add a safe-guard to the line search. In the line search algorithm, when searching for the step length that will decrease the total energy, also check if the new energy value is finite, e.g. with `std::isfinite`, and continue searching, if it is not.

check if energy is finite (not NAN)

Algorithm: line_search
Input: x, dx, α, β
while $E(x - \alpha * dx) > E(x)$ **do**
  $\alpha = \alpha * \beta;$
end do;

Exercise 2.3 (Bonus)

(1) compute exact gradient/Hessian using Automatic Differentiation
(2) compute exact gradient/Hessian by deriving analytical solution by hand, as in exercise 1

Automatic Differentiation

What is it?

Instead of computing derivatives by hand, let a program compute derivatives!

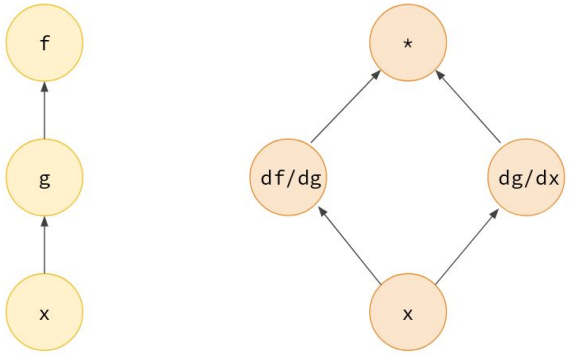
How does it work?

1. chain rule

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$$

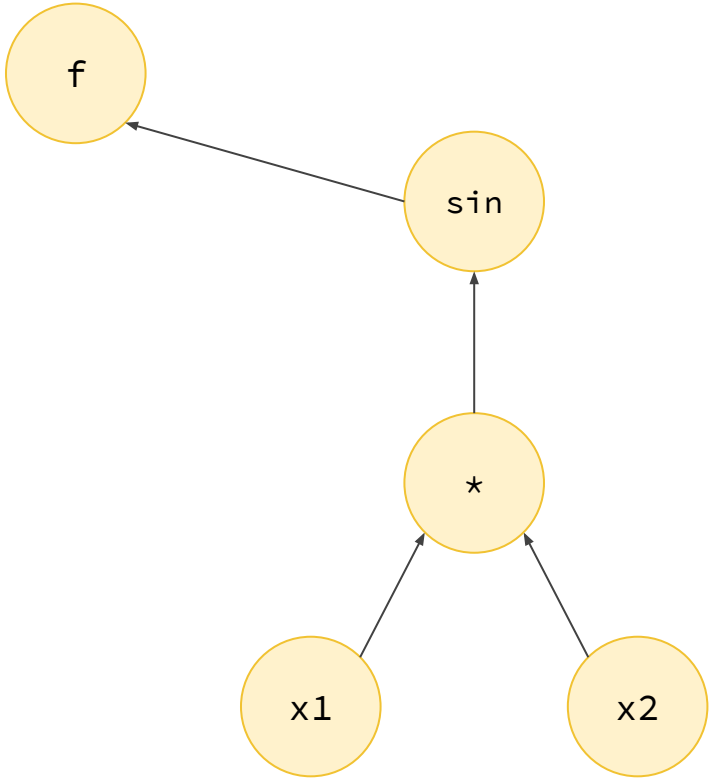
need to know f' need to know chain rule

2. expression graphs



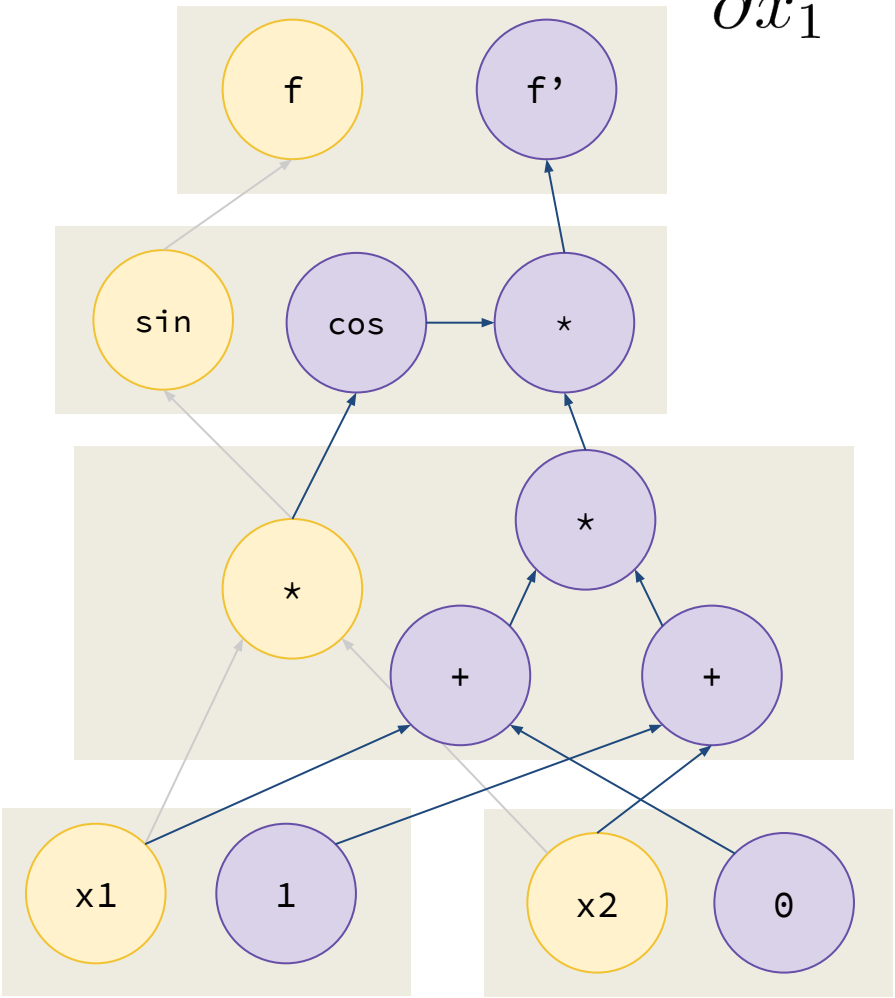
Expression Graphs

$$f(x_1, x_2) = \sin(x_1 x_2)$$

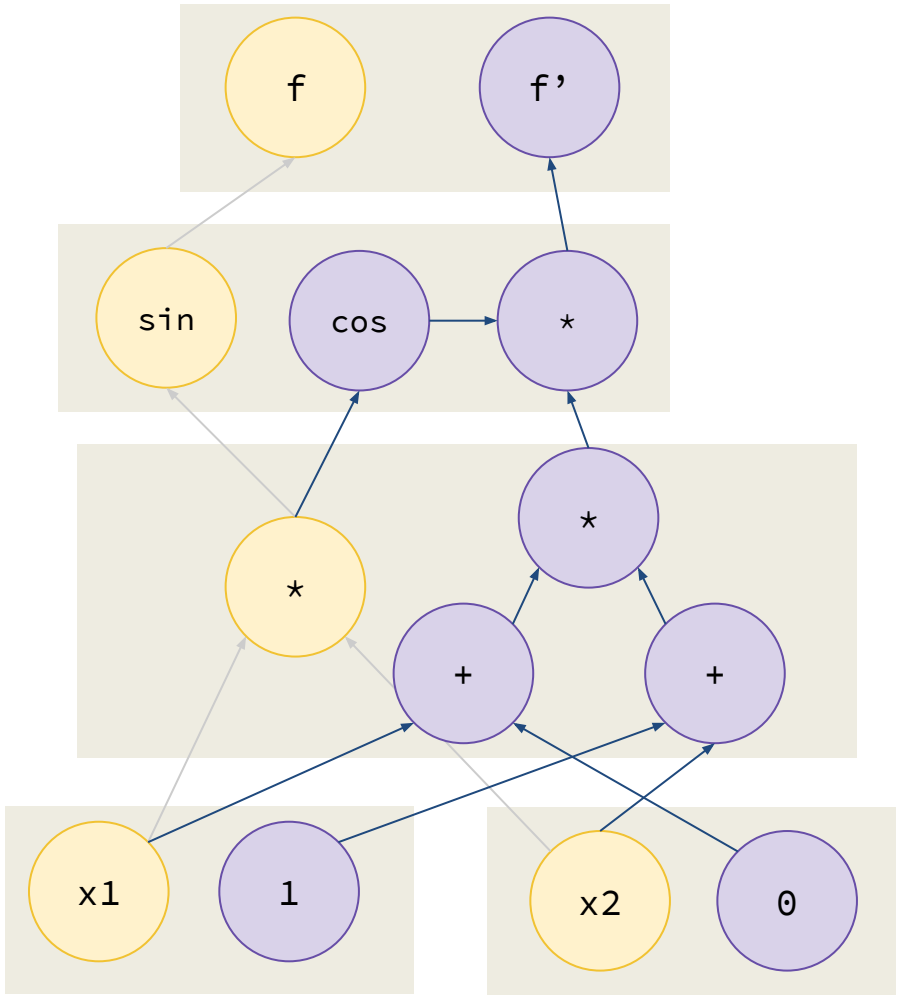


Expression Graphs

$$f(x_1, x_2) = \sin(x_1 x_2) \qquad \frac{\partial f}{\partial x_1} = \cos(x_1 x_2) x_2$$

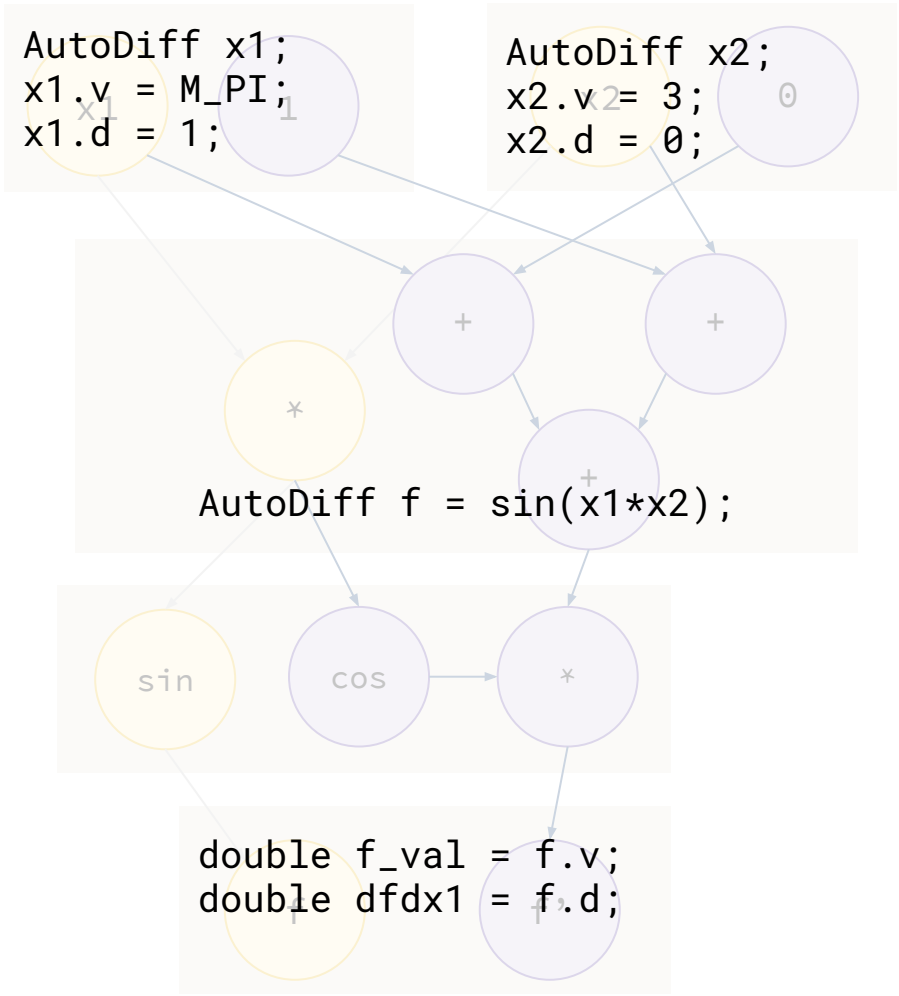


AutoDiff class




```
class AutoDiff {

AutoDiff operator*(AutoDiff b) {
     AutoDiff df;
     df.v = v * b.v;
     df.d = d*b.v + v*b.d;
     return df;
}

     double v, d;
};

AutoDiff sin(AutoDiff b)
{
     AutoDiff df;
     df.v = sin(b.v);
     df.d = cos(b.v) * b.d);
     return df;
}
```

AutoDiff class




```
class AutoDiff {

AutoDiff operator*(AutoDiff b) {
    AutoDiff df;
    df.v = v * b.v;
    df.d = d*b.v + v*b.d;
    return df;
}

    double v, d;
};

AutoDiff sin(AutoDiff b)
{
    AutoDiff df;
    df.v = sin(b.v);
    df.d = cos(b.v) * b.d);
    return df;
}
```

What about higher-order derivatives?

```
// df/dx1
AutoDiff<double> x1; x1.v = M_PI;
x1.d = 1;
AutoDiff<double> x2; x2.v = 3;
x2.d = 0;
AutoDiff<double> f = sin(x1*x2);
double f_val = f.v;
double dfdx1 = f.d;

// ddf/dx1dx2
AutoDiff<AutoDiff<double>> x1;
x1.v.v = M_PI; x1.v.d = 1; x1.d.v = 0;
AutoDiff<AutoDiff<double>> x2;
x2.v.v = 3;    x2.v.d = 0; x2.d.v = 1;
AutoDiff<AutoDiff<double>> f = sin(x1*x2);
double f_val = f.v.v;
double ddfdx1dx2 = f.d.d;

// ddf/dx2dx2
AutoDiff<AutoDiff<double>> x1;
x1.v.v = M_PI; x1.v.d = 0; x1.d.v = 0;
AutoDiff<AutoDiff<double>> x2;
x2.v.v = 3;    x2.v.d = 1; x2.d.v = 1;
AutoDiff<AutoDiff<double>> f = sin(x1*x2);
double f_val = f.v.v;
double ddfdx2dx2 = f.d.d;
```

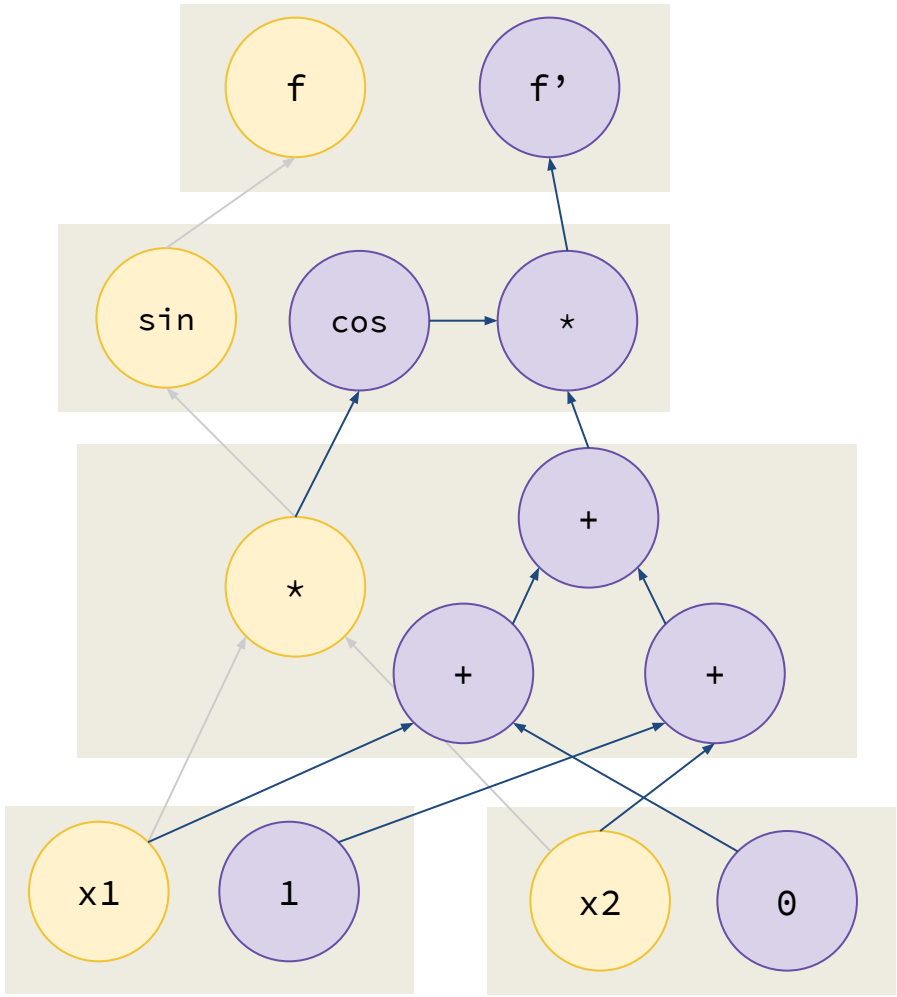
```
template<class T>
class AutoDiff {

    ...

    T v, d;
};
```

Symbolic Differentiation

- Store expression graph
- use algebraic rules to simplify expression graph
- avoid repetition of computation of same operation

  → create hashmap of nodes

- generate code from simplified expression graph
- JIT compilation of generated code



Questions Exercise 1

Do we need to compute the derivative of the function or do we numerically perform the gradient ?

For exercise 1, compute the gradient and hessian analytically, and not numerically with finite differences. The analytical derivatives are in the lecture slides. For exercise 2, see assignment.

Do we need to add (+=) or to replace (=) the calculated values in the gradient Vector ?

You will have to figure this out yourself :)
A hint: In the function `computeSearchDirection`, the vector `dx` has the correct size, but is not guaranteed to be zero in every element!

Questions Exercise 1

Do you have sample/examples of what we should end up with for the results of each question ?

You can check it yourself:
1. is the computed x_min a local minimum of the function? Easy to verify for Rosenbrock function.
2. Is the gradient at x_min zero? (or actually: is it smaller than the residual? In the gradient descent class, variable solveResidual holds this value.)

After clicking "Test", I get a non-zero deformation energy. Is this correct?

We are applying a gravitational force on a body with mass. Thus, the static solution will have a no-zero internal energy. The static solution should however satisfy that the sum over all forces is zero.

Questions Exercise 1

What is the size/structure of x/X/grad/hessian?

In both, the ObjectiveFunction and the Element class, x/X/grad/hessian are global quantities and of size 2N (or 2N x 2N for the hessian).
The structure is: **x**=[x1,y1,x2,y2,...,xn,yn]^T, where (x1,y1) are coordinates of node 1. The same is true for the gradient/hessian.

Typo: The spring energy was missing an L. Corrected pdf is uploaded.

General life advice: If you want fast and good answers, write short and concise emails :)

MacOS + clang → :-/

Good luck!

any questions?

moritzge@inf.ethz.ch

CRL

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich