Differentiating integral type variables in Clad

Petro Zarytskyi

First case: no diff. dependance on input parameters.

```
double f (double x, double y) {
    int step = 2;
    for (int i=0; i < 10; i += step)
        x += y;
    return x;
}
```

```
void f_grad(double x, double y, double *_d_x, double *_d_y)
{
    int _d_step = 0;
    ...
    int _d_i = 0;
    ...
    for (; _t0; _t0--) {
        {
            i = clad::pop(_t1);
            int _r_d0 = _d_i;
            _d_step += _r_d0;
        }
        ...
    }
}
```

Second case: diff. dependance on input parameters.

```
double f (double x) {
    int n = x;
    return n;
}
```

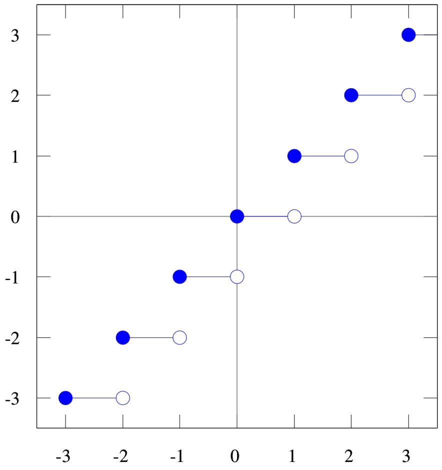
f = floor(x)

```
void f_grad(double x, double *_d_x) {
    int _d_n = 0;
    int n = x;
    goto _label0;
  _label0:
    _d_n += 1;
    *_d_x += _d_n;
}
```

Output: df/dx = 1

Second case: diff. dependance on input parameters.

f = floor(x)



df/dx = 0 when x is not an integer,
undefined otherwise

What do Tapenade and Enzyme do?

- Tapenade doesn’t have adjoints for integral type variables and doesn’t allow lossy assignments (case 2).
- Enzyme doesn’t have adjoints for integral type variables (allows case 2).

Issue 1: Integral type parameters.

```
double f (double x, int y, float z){
    ...
}
```

```
void f_grad(double x, int y, float z,
double *_d_x, float *_d_z) {
    ...
}
```

Issue 1: Integral type parameters.

```
double f (double x, int y, float z){
    ...
}
```

```
void f_grad(double x, int y, float z,
double *_d_x, float *_d_z) {
    ...
}
```

which requires overloads in Clad…

Issue 2: Additional parameters (error estimation)

```
double f (double x, int y, float z){
    ...
}
```

```
void f_grad(double x, int y, float z,
double *_d_x, int *_d_y, float *_d_z,
double& error) {
    ...
}
```

???

Issue 2: Additional parameters (error estimation)

Possible solution 1: force all additional parameters to be of a pointer type

```
double f (double x, int y, float z){
    ...
}
```

```
void f_grad(double x, int y, float z,
double *_d_x, float *_d_z, double* error)
{
    ...
}
```

Issue 2: Additional parameters (error estimation)

Possible solution 2: somehow get rid of overloads?

Pros

- Less useless code.
- Results that are mathematically correct (consistent with numerical differentiation).
- Consistency with other tools like Tapenade and Enzyme.

Cons

- Backward incompatibility with the gradient signature.
- Integral type parameters in error estimation.
- What if the user wants a symbolic derivative with respect to an integer?