Algorithmic Differentiation (AD)

Sample Code Collection*

Uwe Naumann

Computer Science, RWTH Aachen University, Germany
Email: naumann@stce.rwth-aachen.de

Contents

*... for use in Risk Training Masterclass, London, 21-22 March 2018.

1 Primal SDE

Listing 1: Primal SDE

```
#ifndef __F_H_INCLUDED_
#define __F_H_INCLUDED_

#include "std_includes.h"

template<typename AT, typename PT>
void f(AT& x, const vector<AT>& p,
    const vector<vector<PT>>& dW) {
  int m=dW.size(), n=dW[0].size();
  AT s=0, x0=x; PT dt=1./n, t;
  for (int j=0;j<m;j++) {
    t=0;
    for (int i=0;i<n;i++) {
      x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW[j][i];
      t+=dt;
    }
    s+=x; x=x0;
  }
  x=s/m;
}
```

```

#endif
```

Listing 2: Primal SDE (Driver)

```
#include "std_includes.h"
#include "f.h"

int main(int c, char* v[]) {
  assert(c==3);
  int m=atoi(v[1]), n=atoi(v[2]);
  double x=1;
  const vector<double> p(n,1);

  default_random_engine generator;
  normal_distribution<double> distribution(0.0,1.0);
  vector<vector<double>> dW(m,vector<double>(n,1));
  for (int i=0;i<m;i++)
    for (int j=0;j<n;j++)
      dW[i][j]=distribution(generator);

  f(x,p,dW);
  cout << "x=" << x << endl;
  return 0;
}
```

Listing 3: Approximate Tangent SDE

```
#include "std_includes.h"
#include "f.h"

template<typename T>
vector<T> driver(T& x, vector<T>& p,
    const vector<vector<double>>& dW) {
  int n=dW[0].size();
  vector<T> g(n+1,0);
  double x0=x;
  f(x,p,dW);
  double h=sqrt(DBL_EPSILON);
  double xp=x0+h;
  f(xp,p,dW);
  g[0]=(xp-x)/h;
  for (int i=0;i<n;i++) {
    xp=x0; p[i]+=h; f(xp,p,dW); g[i+1]=(xp-x)/h; p[i]-=h;
  }
  return g;
}
```

```
20
21 int main(int c, char* v[]) {
22   assert(c==3); int m=atoi(v[1]), n=atoi(v[2]);
23
24   const double x0=1;
25   vector<double> p(n,1);
26
27   default_random_engine generator;
28   normal_distribution<double> distribution(0.0,1.0);
29   vector<vector<double>> dW(m,vector<double>(n,1));
30   for (int i=0;i<m;i++)
31     for (int j=0;j<n;j++)
32       dW[i][j]=distribution(generator);
33
34   double x=x0;
35   vector<double> g=driver(x,p,dW);
36   cout << "dx/dx0=" << g[0] << endl;
37   for (int i=0;i<n;i++)
38     cout << "dx/dp[" << i << "]=" << g[i+1] << endl;
39   return 0;
40 }
```

2 First-Order AD

2.1 Tangents

Listing 4: Tangent SDE (Handwritten)

```
1  #include "std_includes.h"
2
3  template<typename T>
4  void f_t(T& x, T& xt,
5      const vector<T>& p, vector<T>& pt,
6      const vector<vector<double>>& dW) {
7    int m=dW.size(), n=dW[0].size();
8    T s=0, st=0, x0=x, x0t=xt; double dt=1./n, t;
9    for (int j=0;j<m;j++) {
10     t=0;
11     for (int i=0;i<n;i++) {
12       xt+=dt*sin(x*t)*pt[i]
13           +dt*p[i]*t*cos(x*t)*xt
14           +cos(x*t)*sqrt(dt)*dW[j][i]*pt[i]
15           -p[i]*t*sin(x*t)*sqrt(dt)*dW[j][i]*xt;
16       x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW[j][i];
17       t+=dt;
18     }
```

```
19      st+=xt; s+=x;
20      xt=x0t; x=x0;
21    }
22    xt=st/m; x=s/m;
23  }
24
25  vector<double> driver(double& x, const vector<double>& p,
26      const vector<vector<double>>& dW) {
27    int n=dW[0].size();
28    vector<double> g(n+1,0);
29    double x0=x, xt=1; vector<double> pt(n,0);
30    f_t(x,xt,p,pt,dW);
31    g[0]=xt;
32    for (int i=0;i<n;i++) {
33      x=x0; xt=0; pt[i]=1;
34      f_t(x,xt,p,pt,dW);
35      g[i+1]=xt;
36      pt[i]=0;
37    }
38    return g;
39  }
40
41  int main(int c, char* v[]) {
42    assert(c==3); int m=atoi(v[1]), n=atoi(v[2]);
43
44    const double x0=1;
45    vector<double> p(n,1);
46
47    default_random_engine generator;
48    normal_distribution<double> distribution(0.0,1.0);
49    vector<vector<double>> dW(m,vector<double>(n,1));
50    for (int i=0;i<m;i++)
51      for (int j=0;j<n;j++)
52        dW[i][j]=distribution(generator);
53
54    double x=x0;
55    vector<double> g=driver(x,p,dW);
56    cout << "dx/dx0=" << g[0] << endl;
57    for (int i=0;i<n;i++)
58      cout << "dx/dp[" << i << "]=" << g[i+1] << endl;
59    return 0;
60  }
```

Listing 5: Tangent SDE (dco/c++)

```
1  #include "std_includes.h"
2
```

```
#include "dco.hpp"
typedef dco::gt1s<double>::type DCO_T;

#include "f.h"

vector<double> driver(double& xv, vector<double>& pv,
    const vector<vector<double>>& dW) {
  int n=dW[0].size();
  vector<double> g(n+1,0);
  DCO_T x0=xv;
  vector<DCO_T> p(n); dco::value(p)=pv;
  DCO_T x=x0;
  dco::derivative(x)=1;
  f(x,p,dW);
  g[0]=dco::derivative(x);
  for (int i=0;i<n;i++) {
    x=x0;
    dco::derivative(p[i])=1;
    f(x,p,dW);
    g[i+1]=dco::derivative(x);
    dco::derivative(p[i])=0;
  }
  return g;
}

int main(int c, char* v[]) {
  assert(c==3); int m=atoi(v[1]), n=atoi(v[2]);

  const double x0=1;
  vector<double> p(n,1);

  default_random_engine generator;
  normal_distribution<double> distribution(0.0,1.0);
  vector<vector<double>> dW(m,vector<double>(n,1));
  for (int i=0;i<m;i++)
    for (int j=0;j<n;j++)
      dW[i][j]=distribution(generator);

  double x=x0;
  vector<double> g=driver(x,p,dW);
  cout << "dx/dx0=" << g[0] << endl;
  for (int i=0;i<n;i++)
    cout << "dx/dp[" << i << "]=" << g[i+1] << endl;
  return 0;
}
```

2.2 Adjoints

Listing 6: Adjoint SDE (Handwritten)

```
#include "std_includes.h"

template<typename T>
void f_a(T& x, T& xa, const vector<T>& p, vector<T>& pa,
    const vector<vector<double>>& dW) {
  int m=dW.size(), n=dW[0].size();
  stack<T> tbr_T; stack<double> tbr_double;
  // augmented primal
  T s=0, x0=x; double dt=1./n, t;
  for (int j=0;j<m;j++) {
    t=0;
    for (int i=0;i<n;i++) {
      tbr_T.push(x);
      x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW[j][i];
      tbr_double.push(t);
      t+=dt;
    }
    s+=x; x=x0;
  }
  x=s/m;
  T y=x;
  // adjoint
  T sa=0, x0a=0;
  sa+=xa/m; xa=0;
  for (int j=m-1;j>=0;j--) {
    x0a+=xa; xa=0;
    xa+=sa;
    for (int i=n-1;i>=0;i--) {
      t=tbr_double.top(); tbr_double.pop();
      x=tbr_T.top(); tbr_T.pop();
      pa[i]+=(dt*sin(x*t)+cos(x*t)*sqrt(dt)*dW[j][i])*xa;
      xa=(1+dt*p[i]*t*cos(x*t)-p[i]*t*sin(x*t)*sqrt(dt)*dW[j][i])*xa;
    }
  }
  xa+=x0a; x0a=0;
  x=y;
}

vector<double> driver(double& x, vector<double>& p,
    const vector<vector<double>>& dW) {
  int n=dW[0].size();
  vector<double> g(n+1,0);
```

```
43    double xa=1; vector<double> pa(n,0);
44    f_a(x,xa,p,pa,dW);
45    g[0]=xa;
46    for (int i=0;i<n;i++) g[i+1]=pa[i];
47    return g;
48  }
49
50  int main(int c, char* v[]) {
51    assert(c==3); int m=atoi(v[1]), n=atoi(v[2]);
52    const double x0=1;
53    vector<double> p(n,1);
54    default_random_engine generator;
55    normal_distribution<double> distribution(0.0,1.0);
56    vector<vector<double>> dW(m,vector<double>(n,1));
57    for (int i=0;i<m;i++)
58      for (int j=0;j<n;j++)
59        dW[i][j]=distribution(generator);
60    double x=x0;
61    vector<double> g=driver(x,p,dW);
62    cout << "dx/dx0=" << g[0] << endl;
63    for (int i=0;i<n;i++)
64      cout << "dx/dp[" << i << "]=" << g[i+1] << endl;
65    return 0;
66  }
```

Listing 7: Adjoint SDE (dco/c++)

```
1   #include "std_includes.h"
2
3   #include "dco.hpp"
4   typedef dco::ga1s<double> DCO_AM;
5   typedef DCO_AM::type DCO_A;
6   typedef DCO_AM::tape_t DCO_AM_TAPE;
7
8   #include "f.h"
9
10  vector<double> driver(double& xv, vector<double>& pv,
11      const vector<vector<double>>& dW) {
12    int n=dW[0].size();
13    vector<double> g(n+1,0);
14    DCO_A x0=xv;
15    vector<DCO_A> p(n); dco::value(p)=pv;
16    DCO_AM::global_tape=DCO_AM_TAPE::create();
17    DCO_AM::global_tape->register_variable(x0);
18    DCO_AM::global_tape->register_variable(p);
19    DCO_A x=x0;
20    f(x,p,dW);
```

```
21   DCO_AM::global_tape->register_output_variable(x);
22   dco::derivative(x)=1;
23   DCO_AM::global_tape->interpret_adjoint();
24   g[0]=dco::derivative(x0);
25   for (int i=0;i<n;i++) g[i+1]=dco::derivative(p[i]);
26   DCO_AM_TAPE::remove(DCO_AM::global_tape);
27   return g;
28 }
29
30 int main(int c, char* v[]) {
31   assert(c==3); int m=atoi(v[1]), n=atoi(v[2]);
32
33   const double x0=1;
34   vector<double> p(n,1);
35
36   default_random_engine generator;
37   normal_distribution<double> distribution(0.0,1.0);
38   vector<vector<double>> dW(m,vector<double>(n,1));
39   for (int i=0;i<m;i++)
40     for (int j=0;j<n;j++)
41       dW[i][j]=distribution(generator);
42
43   double x=x0;
44   vector<double> g=driver(x,p,dW);
45   cout << "dx/dx0=" << g[0] << endl;
46   for (int i=0;i<n;i++)
47     cout << "dx/dp[" << i << "]=" << g[i+1] << endl;
48   return 0;
49 }
```

2.3 Improvements

2.3.1 Vector Tangents

See LIBOR.

2.3.2 Pathwise Ajoints

Listing 8: Adjoint SDE: Pathwise Adjoints (Handwritten)

```
1 #include "std_includes.h"
2
3 enum Mode { PRIMAL, CONTEXT_FREE_JOINT_ADJOINT };
4
5 void path(Mode mode, const int n,
6   double& x, double& xa,
7   const vector<double>& p, vector<double>& pa,
```

```
8     const vector<double>& dW_j) {
9     double t=0, dt=1.0/n;
10    switch (mode) {
11      case PRIMAL:
12        for (int i=0;i<n;i++) {
13          x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW_j[i];
14          t+=dt;
15        }
16        break;
17      case CONTEXT_FREE_JOINT_ADJOINT:
18        stack<double> tbr;
19        // augmented primal
20        t=0;
21        for (int i=0;i<n;i++) {
22          tbr.push(x);
23          x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW_j[i];
24          t+=dt;
25        }
26        // adjoint
27        t=1;
28        for (int i=n-1;i>=0;i--) {
29          t-=dt;
30          x=tbr.top(); tbr.pop();
31          pa[i]+=(dt*sin(x*t)+cos(x*t)*sqrt(dt)*dW_j[i])*xa;
32          xa=(1+dt*p[i]*t*cos(x*t)-p[i]*t*sin(x*t)*sqrt(dt)*dW_j[i])*xa;
33        }
34    }
35  }
36
37  void f_a(double& x, double& xa,
38      const vector<double>& p, vector<double>& pa,
39      const vector<vector<double>>& dW) {
40    int m=dW.size(), n=dW[0].size();
41    // augmented primal
42    double s=0, x0=x;
43    for (int j=0;j<m;j++) {
44      x=x0;
45      path(PRIMAL,n,x,xa,p,pa,dW[j]);
46      s+=x;
47    }
48    x=s/m;
49    double y=x;
50    // adjoint
51    double sa=0,x0a=0;
52    sa+=xa/m; xa=0;
53    for (int j=m-1;j>=0;j--) {
```

```
54      x=x0; xa+=sa;
55      path(CONTEXT_FREE_JOINT_ADJOINT,n,x,xa,p,pa,dW[j]);
56      x0a+=xa; xa=0;
57    }
58    xa+=x0a; x0a=0;
59    x=y;
60  }
61
62  vector<double> driver(double& x, vector<double>& p,
63      const vector<vector<double>>& dW) {
64    int n=dW[0].size();
65    vector<double> g(n+1,0);
66    double xa=1; vector<double> pa(n,0);
67    f_a(x,xa,p,pa,dW);
68    g[0]=xa;
69    for (int i=0;i<n;i++) g[i+1]=pa[i];
70    return g;
71  }
72
73  int main(int c, char* v[]) {
74    assert(c==3); int m=atoi(v[1]), n=atoi(v[2]);
75    const double x0=1;
76    vector<double> p(n,1);
77    default_random_engine generator;
78    normal_distribution<double> distribution(0.0,1.0);
79    vector<vector<double>> dW(m,vector<double>(n,1));
80    for (int i=0;i<m;i++)
81      for (int j=0;j<n;j++)
82        dW[i][j]=distribution(generator);
83    double x=x0;
84    vector<double> g=driver(x,p,dW);
85    cout << "dx/dx0=" << g[0] << endl;
86    for (int i=0;i<n;i++)
87      cout << "dx/dp[" << i << "]=" << g[i+1] << endl;
88    return 0;
89  }
```

2.3.3 Preaccumulation

Listing 9: Adjoint SDE: Preaccumulation (dco/c++)

```
1  #include "std_includes.h"
2
3  #include "dco.hpp"
4  typedef dco::ga1s<double> DCO_AM;
5  typedef DCO_AM::type DCO_A;
6  typedef DCO_AM::tape_t DCO_AM_TAPE;
```

```
7
8  template<typename AT, typename PT>
9  void f(AT& x, const vector<AT>& p, const vector<vector<PT>>& dW) {
10   int m=dW.size(), n=dW[0].size();
11   AT s=0, x0=x; PT dt=1./n, t;
12   for (int j=0;j<m;j++) {
13     DCO_AM::jacobian_preaccumulator_t jp(dco::tape(x));
14     t=0;
15     jp.start();
16     for (int i=0;i<n;i++) {
17       x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW[j][i];
18       t+=dt;
19     }
20     jp.register_output(x);
21     jp.finish();
22     s+=x; x=x0;
23   }
24   x=s/m;
25 }
26
27 vector<double> driver(double& xv, vector<double>& pv,
28     const vector<vector<double>>& dW) {
29   int n=dW[0].size();
30   vector<double> g(n+1,0);
31   DCO_A x0=xv;
32   vector<DCO_A> p(n); dco::value(p)=pv;
33   DCO_AM::global_tape=DCO_AM_TAPE::create();
34   DCO_AM::global_tape->register_variable(x0);
35   DCO_AM::global_tape->register_variable(p);
36   DCO_A x=x0;
37   f(x,p,dW);
38   DCO_AM::global_tape->register_output_variable(x);
39   dco::derivative(x)=1;
40   DCO_AM::global_tape->interpret_adjoint();
41   g[0]=dco::derivative(x0);
42   for (int i=0;i<n;i++) g[i+1]=dco::derivative(p[i]);
43   DCO_AM_TAPE::remove(DCO_AM::global_tape);
44   return g;
45 }
46
47 int main(int c, char* v[]) {
48   assert(c==3); int m=atoi(v[1]), n=atoi(v[2]);
49
50   const double x0=1;
51   vector<double> p(n,1);
52
```

```
53   default_random_engine generator;
54   normal_distribution<double> distribution(0.0,1.0);
55   vector<vector<double>> dW(m,vector<double>(n,1));
56   for (int i=0;i<m;i++)
57     for (int j=0;j<n;j++)
58       dW[i][j]=distribution(generator);
59
60   double x=x0;
61   vector<double> g=driver(x,p,dW);
62   cout << "dx/dx0=" << g[0] << endl;
63   for (int i=0;i<n;i++)
64     cout << "dx/dp[" << i << "]=" << g[i+1] << endl;
65   return 0;
66 }
```

3 Second-Order AD

3.1 Tangents

Listing 10: Second-Order Tangent SDE (dco/c++)

```
1  #include "std_includes.h"
2
3  #include "dco.hpp"
4  typedef dco::gt1s<double>::type DCO_T;
5  typedef dco::gt1s<DCO_T>::type DCO_TT;
6
7  #include "f.h"
8
9  vector<vector<double>> driver(
10     double& xv, const vector<double> &pv,
11     const vector<vector<double>>& dW) {
12   int n=pv.size();
13   vector<DCO_TT> p(n); dco::passive_value(p)=pv;
14   vector<vector<double>> ddxdpp(n,vector<double>(n,0));
15   for (int i=0;i<n;i++) {
16     dco::derivative(dco::value(p[i]))=1;
17     for (int j=0;j<=i;j++) {
18       dco::value(dco::derivative(p[j]))=1;
19       DCO_TT x=xv;
20       f(x,p,dW);
21       ddxdpp[i][j]=dco::derivative(dco::derivative(x));
22       dco::value(dco::derivative(p[j]))=0;
23     }
24     dco::derivative(dco::value(p[i]))=0;
25   }
```

```
26   return ddxdpp;
27 }
28
29 int main(int c, char* v[]) {
30   assert(c==3);
31   int m=atoi(v[1]), n=atoi(v[2]);
32
33   double x=1;
34   vector<double> p(n,1);
35
36   default_random_engine generator;
37   normal_distribution<double> distribution(0.0,1.0);
38   vector<vector<double>> dW(m,vector<double>(n,1));
39   for (int i=0;i<m;i++)
40     for (int j=0;j<n;j++)
41       dW[i][j]=distribution(generator);
42
43   vector<vector<double>> ddxdpp=driver(x,p,dW);
44   for (int i=0;i<n;i++)
45     for (int j=0;j<=i;j++)
46       cout << "ddx/dpp[" << i << "][" << j << "]="
47           << ddxdpp[i][j] << endl;
48   return 0;
49 }
```

3.2 Adjoints

Listing 11: Second-Order Adjoint SDE (dco/c++)

```
1  #include "std_includes.h"
2
3  #include "dco.hpp"
4  typedef dco::gt1s<double>::type DCO_T;
5  typedef dco::ga1s<DCO_T> DCO_TAM;
6  typedef DCO_TAM::type DCO_TA;
7  typedef DCO_TAM::tape_t DCO_TAM_TAPE;
8  typedef DCO_TAM_TAPE::position_t DCO_TAM_TAPE_POS;
9
10 #include "f.h"
11
12 vector<vector<double>> driver(
13     double& xv, const vector<double> &pv,
14     const vector<vector<double>>& dW) {
15   int n=pv.size();
16   vector<DCO_TA> p(n); dco::passive_value(p)=pv;
17   vector<vector<double>> ddxdpp(n,vector<double>(n,0));
18   DCO_TAM::global_tape=DCO_TAM_TAPE::create();
```

```
19    DCO_TAM::global_tape->register_variable(p);
20    DCO_TAM_TAPE_POS tpos=DCO_TAM::global_tape->get_position();
21    for (int i=0;i<n;i++) {
22      dco::derivative(dco::value(p[i]))=1;
23      DCO_TA x=xv;
24      f(x,p,dW);
25      dco::value(dco::derivative(x))=1;
26      DCO_TAM::global_tape->interpret_adjoint_and_reset_to(tpos);
27      for (int j=0;j<=i;j++)
28        ddxdpp[i][j]=dco::derivative(dco::derivative(p[j]));
29      for (int j=0;j<n;j++) {
30        dco::derivative(dco::derivative(p[j]))=0;
31        dco::value(dco::derivative(p[j]))=0;
32      }
33      dco::derivative(dco::value(p[i]))=0;
34    }
35    DCO_TAM_TAPE::remove(DCO_TAM::global_tape);
36    return ddxdpp;
37  }
38
39  int main(int c, char* v[]) {
40    assert(c==3);
41    int m=atoi(v[1]), n=atoi(v[2]);
42
43    double x=1;
44    vector<double> p(n,1);
45
46    default_random_engine generator;
47    normal_distribution<double> distribution(0.0,1.0);
48    vector<vector<double>> dW(m,vector<double>(n,1));
49    for (int i=0;i<m;i++)
50      for (int j=0;j<n;j++)
51        dW[i][j]=distribution(generator);
52
53    vector<vector<double>> ddxdpp=driver(x,p,dW);
54    for (int i=0;i<n;i++)
55      for (int j=0;j<=i;j++)
56        cout << "ddx/dpp[" << i << "][" << j << "]="
57            << ddxdpp[i][j] << endl;
58    return 0;
59  }
```

4 Beyond Black-Box AD

4.1 Implicit Functions

4.1.1 Tangents

Listing 12: Algorithmic Tangent Nonlinear Equation (Handwritten)

```
#include "std_includes.h"

template<typename T>
void f_t(T& xv, T& xt, const T& pv, const T& pt, const T& eps) {
  while (abs(xv*xv-pv)>eps) {
    xt+=pt/(2*xv)-(3./4.+pv/(4*xv*xv))*xt;
    xv-=(xv*xv-pv)/(2*xv);
  }
}

int main(int c, char* v[]) {
  assert(c==2);
  double pv=atof(v[1]), xv=1;
  double pt=1, xt=0;
  const double eps=1e-12;
  f_t(xv,xt,pv,pt,eps);
  cout << "x=" << xv << endl;
  cout << "dxdp=" << xt << endl;
  return 0;
}
```

Listing 13: Symbolic Tangent Nonlinear Equation (Handwritten)

```
#include "std_includes.h"

template<typename T>
void f(T& x, const T& p, const T& eps) {
  while (abs(x*x-p)>eps) x=x-(x*x-p)/(2*x);
}

template<typename T>
void f_st(const T& xv, T& xt, const T& pt) {
  xt=pt/(2*xv);
}

int main(int c, char* v[]) {
  assert(c==2);
  double pv=atof(v[1]), xv=1;
  double pt=1, xt=0;
```

```
17   const double eps=1e-12;
18   f(xv,pv,eps);
19   f_st(xv,xt,pt);
20   cout << "x=" << xv << endl;
21   cout << "dxdp=" << xt << endl;
22   return 0;
23 }
```

4.1.2 Adjoints

Listing 14: Algorithmic Adjoint Nonlinear Equation (Handwritten)

```
1  #include "std_includes.h"
2
3  template<typename T>
4  void f_a(T& xv, T& xa, const T& pv, T& pa, const T& eps) {
5    stack<T> tbr_T;
6    int i=0;
7    while (abs(xv*xv-pv)>eps) {
8      tbr_T.push(xv);
9      xv-=(xv*xv-pv)/(2*xv);
10     i++;
11   }
12   double y=xv;
13   for (int j=0;j<i;j++) {
14     xv=tbr_T.top(); tbr_T.pop();
15     pa+=xa/(2*xv);
16     xa-=(3./4.+pv/(4*xv*xv))*xa;
17   }
18   xv=y;
19 }
20
21 int main(int c, char* v[]) {
22   assert(c==2);
23   double pv=atof(v[1]), xv=1;
24   double pa=0, xa=1;
25   const double eps=1e-12;
26   f_a(xv,xa,pv,pa,eps);
27   cout << "x=" << xv << endl;
28   cout << "dxdp=" << pa << endl;
29   return 0;
30 }
```

Listing 15: Symbolic Adjoint Nonlinear Equation (Handwritten)

```
1  #include "std_includes.h"
2
```

```
template<typename T>
void f(T& x, const T& p, const T& eps) {
  while (abs(x*x-p)>eps) x=x-(x*x-p)/(2*x);
}

template<typename T>
void f_sa(const T& xv, T& xa, T& pa) {
  pa+=xa/(2*xv); xa=0;
}

int main(int c, char* v[]) {
  assert(c==2);
  double pv=atof(v[1]), xv=1;
  double pa=0, xa=1;
  const double eps=1e-12;
  f(xv,pv,eps);
  f_sa(xv,xa,pa);
  cout << "x=" << xv << endl;
  cout << "dxdp=" << pa << endl;
  return 0;
}
```

4.2 Checkpointing

Listing 16: Adjoint SDE: Pathwise Adjoints with Equidistant Checkpointing (Handwritten)

```
#include "std_includes.h"

enum Mode { PRIMAL, CONTEXT_FREE_JOINT_FORWARD, CONTEXT_FREE_JOINT_BACKWARD,
            CONTEXT_SENSITIVE_JOINT };

template<typename T>
void steps(Mode mode, int from, int to, T& x, T &xa,
           const vector<T>& p, vector<T>& pa,
           const vector<double>& dW_j) {
  static stack<T> tbr_T; static stack<double> tbr_d;
  int n=p.size(); double dt=1.0/n, t=from*dt;
  switch (mode) {
    default: assert(false); break;
    case CONTEXT_FREE_JOINT_FORWARD:
      tbr_T.push(x); tbr_d.push(t);
      for (int i=from;i<to;i++) {
        x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW_j[i];
        t+=dt;
      }
      break;
```

```
21      case CONTEXT_FREE_JOINT_BACKWARD:
22        t=tbr_d.top(); tbr_d.pop(); x=tbr_T.top(); tbr_T.pop();
23      case CONTEXT_SENSITIVE_JOINT:
24        for (int i=from;i<to;i++) {
25          tbr_T.push(x);
26          x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW_j[i];
27          t+=dt;
28        }
29        double y=x;
30        for (int i=to-1;i>=from;i--) {
31          t-=dt;
32          x=tbr_T.top(); tbr_T.pop();
33          pa[i]+=(dt*sin(x*t)+cos(x*t)*sqrt(dt)*dW_j[i])*xa;
34          xa=(1+dt*p[i]*t*cos(x*t)-p[i]*t*sin(x*t)*sqrt(dt)*dW_j[i])*xa;
35        }
36        x=y;
37    }
38  }
39
40  template<typename T>
41  void path(Mode mode, const int ncs,
42    T& x, T& xa, const vector<T>& p, vector<T>& pa,
43    const vector<double>& dW_j) {
44    int n=dW_j.size();
45    double t=0, dt=1.0/n;
46    switch (mode) {
47      default: assert(false); break;
48      case PRIMAL:
49        for (int i=0;i<n;i++) {
50          x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW_j[i];
51          t+=dt;
52        }
53        break;
54      case CONTEXT_SENSITIVE_JOINT:
55        t=0;
56        for (int i=0;i<n-ncs;i+=ncs)
57          steps(CONTEXT_FREE_JOINT_FORWARD,i,i+ncs,x,xa,p,pa,dW_j);
58        steps(CONTEXT_SENSITIVE_JOINT,n-ncs,n,x,xa,p,pa,dW_j);
59        T y=x;
60        for (int i=n-2*ncs;i>=0;i-=ncs)
61          steps(CONTEXT_FREE_JOINT_BACKWARD,i,i+ncs,x,xa,p,pa,dW_j);
62        x=y;
63    }
64  }
65
66  void f_a(const int ncs, double& x, double& xa,
```

```
67      const vector<double>& p, vector<double>& pa,
68      const vector<vector<double>>& dW) {
69    int m=dW.size();
70    // augmented primal
71    double s=0, x0=x;
72    for (int j=0;j<m;j++) {
73      x=x0;
74      path(PRIMAL,ncs,x,xa,p,pa,dW[j]);
75      s+=x;
76    }
77    x=s/m;
78    double y=x;
79    // adjoint
80    double sa=0,x0a=0;
81    sa+=xa/m; xa=0;
82    for (int j=m-1;j>=0;j--) {
83      x=x0; xa+=sa;
84      path(CONTEXT_SENSITIVE_JOINT,ncs,x,xa,p,pa,dW[j]);
85      x0a+=xa; xa=0;
86    }
87    xa+=x0a; x0a=0;
88    x=y;
89  }
90
91  vector<double> driver(const int ncs, double& x, vector<double>& p,
92      const vector<vector<double>>& dW) {
93    int n=dW[0].size();
94    vector<double> g(n+1,0);
95    double xa=1; vector<double> pa(n,0);
96    f_a(ncs,x,xa,p,pa,dW);
97    g[0]=xa;
98    for (int i=0;i<n;i++) g[i+1]=pa[i];
99    return g;
100 }
101
102 int main(int c, char* v[]) {
103   assert(c==4); int m=atoi(v[1]), n=atoi(v[2]), ncs=atoi(v[3]);
104   const double x0=1;
105   vector<double> p(n,1);
106   default_random_engine generator;
107   normal_distribution<double> distribution(0.0,1.0);
108   vector<vector<double>> dW(m,vector<double>(n,1));
109   for (int i=0;i<m;i++)
110     for (int j=0;j<n;j++)
111       dW[i][j]=distribution(generator);
112   double x=x0;
```

```
  vector<double> g=driver(ncs,x,p,dW);
  cout << "dx/dx0=" << g[0] << endl;
  for (int i=0;i<n;i++)
    cout << "dx/dp[" << i << "]=" << g[i+1] << endl;
  return 0;
}
```

A PDE / Explicit Scheme

Listing 17: Primal PDE / Explicit Scheme

```
#ifndef __F_H_INCLUDED_
#define __F_H_INCLUDED_

template <typename AT, typename PT>
inline void step(const int m, const vector<PT>& p, vector<AT>& y) {
  int n=y.size();
  vector<AT> r(n);
  AT v=p[0]*(n+1)*(n+1);
  r[0]=v*(p[1]-2*y[0]+y[1]);
  for (int i=1;i<n-1;i++) r[i]=v*(y[i-1]-2*y[i]+y[i+1]);
  r[n-1]=v*(y[n-2]-2*y[n-1]+p[2]);
  for (int i=0;i<n;i++) y[i]+=r[i]/m;
}

template <typename AT, typename PT>
inline void f(const int m, const vector<PT>& p, vector<AT>& y) {
  for (int j=0;j<m;j++) step(m,p,y);
}

#endif
```

Listing 18: Primal PDE / Explicit Scheme (Driver)

```
#include "std_includes.h"

#include "f.h"

int main(int c, char* v[]){
  assert(c==3);
  int n=atoi(v[1]), m=atoi(v[2]);
  vector<double> y(n), p={1e-3,42,0};
  for (int i=0;i<n;i++) y[i]=(i+1)*log(static_cast<double>(i+2));
  f(m,p,y);
  cout << 0 << " " << p[1] << endl;
  for (int i=0;i<n;i++)
    cout << static_cast<double>(i+1)/(n+1) << " " << y[i] << endl;
```

```
14    cout << 1 << " " << p[2] << endl;
15    return 0;
16  }
```

A.1 Tangents

Listing 19: Tangent PDE / Explicit Scheme (Handwritten)

```
1   #include "std_includes.h"
2
3   template <typename AT, typename PT>
4   inline void step_t(const int m,
5     const vector<PT>& p, const vector<PT>& p_t,
6     vector<AT>& y, vector<AT>& y_t)
7   {
8     int n=y.size();
9     vector<AT> r(n), r_t(n);
10    int ns=(n+1)*(n+1);
11    AT v=p[0]*ns;
12    r_t[0]=p_t[0]*ns*(p[1]-2*y[0]+y[1])
13          +v*p_t[1]-v*2*y_t[0]+v*y_t[1];
14    r[0]=v*(p[1]-2*y[0]+y[1]);
15    for (int i=1;i<n-1;i++) {
16      r_t[i]=p_t[0]*ns*(y[i-1]-2*y[i]+y[i+1])
17            +v*y_t[i-1]-v*2*y_t[i]+v*y_t[i+1];
18      r[i]=v*(y[i-1]-2*y[i]+y[i+1]);
19    }
20    r_t[n-1]=p_t[0]*ns*(y[n-2]-2*y[n-1]+p[2])
21            +v*y_t[n-2]-v*2*y_t[n-1]+v*p_t[2];
22    r[n-1]=v*(y[n-2]-2*y[n-1]+p[2]);
23    for (int i=0;i<n;i++) {
24      y_t[i]+=r_t[i]/m;
25      y[i]+=r[i]/m;
26    }
27  }
28
29  template <typename AT, typename PT>
30  inline void f_t(const int m,
31    const vector<PT>& p, const vector<PT>& p_t,
32    vector<AT>& y, vector<AT>& y_t)
33  {
34    for (int j=0;j<m;j++) step_t(m,p,p_t,y,y_t);
35  }
36
37  int main(int c, char* v[]) {
38    cout.precision(15);
39    assert(c==3);
```

```
40   int n=atoi(v[1]), m=atoi(v[2]);
41   vector<double> y(n), y_t(n);
42   vector<double> p={1e-3,42,0}, p_t(3,0);
43   for (int j=0;j<n;j++) {
44     for (int i=0;i<n;i++) {
45       y[i]=(i+1)*log(static_cast<double>(i+2));
46       y_t[i]=0;
47     }
48     y_t[j]=1;
49     f_t(m,p,p_t,y,y_t);
50     cout << "dy(n/2)/dy0[" << j << "]=" << y_t[n/2] << endl;
51   }
52   return 0;
53 }
```

Listing 20: Tangent PDE / Explicit Scheme (dco/c++)

```
1  #include "std_includes.h"
2
3  #include "dco.hpp"
4  typedef dco::gt1s<double>::type DCO_T;
5
6  #include "f.h"
7
8  int main(int argc, char* argv[]){
9    assert(argc==3);
10   int n=atoi(argv[1]), m=atoi(argv[2]);
11   vector<DCO_T> y(n), p={1e-3,42,0};
12   for (int j=0;j<n;j++) {
13     for (int i=0;i<n;i++) y[i]=(i+1)*log(static_cast<double>(i+2));
14     dco::derivative(y[j])=1;
15     f(m,p,y);
16     cout << "dy(n/2)/dy0[" << j << "]=" << dco::derivative(y[n/2]) << endl;
17   }
18   return 0;
19 }
```

A.2 Adjoints

Listing 21: Adjoint PDE / Explicit Scheme (Handwritten)

```
1  #include "std_includes.h"
2
3  enum Mode { AUGMENTED_PRIMAL, SPLIT_ADJOINT };
4
5  template <typename AT, typename PT>
6  inline void step_a(Mode mode, const int m,
```

```
7     const vector<PT>& p, vector<PT>& p_a,
8     vector<AT>& y, vector<AT>& y_a)
9   {
10    int n=y.size();
11    static stack<vector<AT>> tbr;
12    vector<AT> r(n), r_a(n,0);
13    int ns=(n+1)*(n+1); AT v=p[0]*ns;
14    switch (mode) {
15    case AUGMENTED_PRIMAL:
16      r[0]=v*(p[1]-2*y[0]+y[1]);
17      for (int i=1;i<n-1;i++)
18        r[i]=v*(y[i-1]-2*y[i]+y[i+1]);
19      r[n-1]=v*(y[n-2]-2*y[n-1]+p[2]);
20      tbr.push(y);
21      for (int i=0;i<n;i++) y[i]+=r[i]/m;
22      break;
23    case SPLIT_ADJOINT:
24      y=tbr.top(); tbr.pop();
25      for (int i=0;i<n;i++) r_a[i]+=y_a[i]/m;
26      p_a[0]+=ns*(p[1]-2*y[0]+y[1])*r_a[0];
27      p_a[1]+=v*r_a[0]; y_a[0]-=v*2*r_a[0];
28      y_a[1]+=v*r_a[0]; r_a[0]=0;
29      for (int i=1;i<n-1;i++) {
30        p_a[0]+=ns*(y[i-1]-2*y[i]+y[i+1])*r_a[i];
31        y_a[i-1]+=v*r_a[i]; y_a[i]-=v*2*r_a[i];
32        y_a[i+1]+=v*r_a[i]; r_a[i]=0;
33      }
34      p_a[0]+=ns*(y[n-2]-2*y[n-1]+p[2])*r_a[n-1];
35      y_a[n-2]+=v*r_a[n-1]; y_a[n-1]-=v*2*r_a[n-1];
36      p_a[2]+=v*r_a[n-1]; r_a[n-1]=0;
37      break;
38    }
39  }
40
41  template <typename AT, typename PT>
42  inline void f_a(const int m,
43    const vector<PT>& p, vector<PT>& p_a,
44    vector<AT>& y, vector<AT>& y_a)
45  {
46    for (int j=0;j<m;j++) step_a(AUGMENTED_PRIMAL,m,p,p_a,y,y_a);
47    for (int j=0;j<m;j++) step_a(SPLIT_ADJOINT,m,p,p_a,y,y_a);
48  }
49
50  int main(int c, char* v[]) {
51    cout.precision(15);
52    assert(c==3);
```

```
53   int n=atoi(v[1]), m=atoi(v[2]);
54   vector<double> y(n), y_a(n,0);
55   for (int i=0;i<n;i++) y[i]=(i+1)*log(static_cast<double>(i+2));
56   vector<double> p={1e-3,42,0}, p_a(3,0);
57   y_a[n/2]=1;
58   f_a(m,p,p_a,y,y_a);
59   for (int i=0;i<n;i++)
60     cout << "dy(n/2)/dy0[" << i << "]=" << y_a[i] << endl;
61   return 0;
62 }
```

Listing 22: Adjoint PDE / Explicit Scheme (dco/c++)

```
1  #include "std_includes.h"
2
3  #include "dco.hpp"
4  typedef double DCO_BT;
5  typedef dco::ga1sm<DCO_BT> DCO_AM;
6  typedef DCO_AM::type DCO_A;
7  typedef DCO_AM::tape_t DCO_AM_TAPE;
8
9  #include "f.h"
10
11 int main(int c, char* v[]){
12   assert(c==3); int n=atoi(v[1]), m=atoi(v[2]);
13   vector<DCO_A> y0(n), p={1e-3,42,0};
14   for (int i=0;i<n;i++) y0[i]=(i+1)*log(static_cast<double>(i+2));
15   DCO_AM_TAPE* tape=DCO_AM_TAPE::create();
16   tape->register_variable(y0);
17   vector<DCO_A> y=y0;
18   f(m,p,y);
19   tape->register_output_variable(y);
20   dco::derivative(y[n/2])=1.;
21   tape->interpret_adjoint();
22   for(int i=0;i<n;i++)
23     cout << "dy(n/2)/dy0[" << i << "]=" << dco::derivative(y0[i]) << endl;
24   DCO_AM_TAPE::remove(tape);
25   return 0;
26 }
```

Listing 23: Adjoint PDE / Explicit Scheme with Equidistant Checkpointing (Handwritten)

```
1  #include "std_includes.h"
2
3  #include "Eigen/Dense"
4  using namespace Eigen;
```

```
5
6  enum Mode { PRIMAL, AUGMENTED_PRIMAL, SPLIT_ADJOINT };
7
8  template <typename AT, typename PT>
9  inline void step_a(Mode mode, const int m,
10   const vector<PT>& p, vector<PT>& p_a,
11   vector<AT>& y, vector<AT>& y_a)
12 {
13   int n=y.size();
14   static stack<vector<AT>> tbr;
15   vector<AT> r(n), r_a(n,0);
16   int ns=(n+1)*(n+1); AT v=p[0]*ns;
17   switch (mode) {
18   case PRIMAL:
19     r[0]=v*(p[1]-2*y[0]+y[1]);
20     for (int i=1;i<n-1;i++)
21       r[i]=v*(y[i-1]-2*y[i]+y[i+1]);
22     r[n-1]=v*(y[n-2]-2*y[n-1]+p[2]);
23     for (int i=0;i<n;i++) y[i]+=r[i]/m;
24     break;
25   case AUGMENTED_PRIMAL:
26     r[0]=v*(p[1]-2*y[0]+y[1]);
27     for (int i=1;i<n-1;i++)
28       r[i]=v*(y[i-1]-2*y[i]+y[i+1]);
29     r[n-1]=v*(y[n-2]-2*y[n-1]+p[2]);
30     tbr.push(y);
31     for (int i=0;i<n;i++) y[i]+=r[i]/m;
32     break;
33   case SPLIT_ADJOINT:
34     y=tbr.top(); tbr.pop();
35     for (int i=0;i<n;i++) r_a[i]+=y_a[i]/m;
36     p_a[0]+=ns*(p[1]-2*y[0]+y[1])*r_a[0];
37     p_a[1]+=v*r_a[0]; y_a[0]-=v*2*r_a[0];
38     y_a[1]+=v*r_a[0]; r_a[0]=0;
39     for (int i=1;i<n-1;i++) {
40       p_a[0]+=ns*(y[i-1]-2*y[i]+y[i+1])*r_a[i];
41       y_a[i-1]+=v*r_a[i]; y_a[i]-=v*2*r_a[i];
42       y_a[i+1]+=v*r_a[i]; r_a[i]=0;
43     }
44     p_a[0]+=ns*(y[n-2]-2*y[n-1]+p[2])*r_a[n-1];
45     y_a[n-2]+=v*r_a[n-1]; y_a[n-1]-=v*2*r_a[n-1];
46     p_a[2]+=v*r_a[n-1]; r_a[n-1]=0;
47     break;
48   }
49 }
50
```

```
template <typename AT, typename PT>
inline void f_a(const int m, const int ncs,
  const vector<PT>& p, vector<PT>& p_a,
  vector<AT>& y, vector<AT>& y_a)
{
  stack<vector<AT>> cp;
  for (int j=0;j<m-ncs;j+=ncs) {
    cp.push(y);
    for (int i=0;i<ncs;i++)
      step_a(PRIMAL,m,p,p_a,y,y_a);
  }
  for (int i=0;i<ncs;i++)
    step_a(AUGMENTED_PRIMAL,m,p,p_a,y,y_a);
  for (int i=0;i<ncs;i++)
    step_a(SPLIT_ADJOINT,m,p,p_a,y,y_a);
  for (int j=0;j<m-ncs;j+=ncs) {
    y=cp.top(); cp.pop();
    for (int i=0;i<ncs;i++)
      step_a(AUGMENTED_PRIMAL,m,p,p_a,y,y_a);
    for (int i=0;i<ncs;i++)
      step_a(SPLIT_ADJOINT,m,p,p_a,y,y_a);
  }
}

int main(int c, char* v[]) {
  assert(c==4);
  int n=atoi(v[1]), m=atoi(v[2]), ncs=atoi(v[3]);
  vector<double> y(n), y_a(n,0);
  for (int i=0;i<n;i++) y[i]=(i+1)*log(static_cast<double>(i+2));
  vector<double> p={1e-3,42,0}, p_a(3,0);
  y_a[n/2]=1;
  f_a(m,ncs,p,p_a,y,y_a);
  for (int i=0;i<n;i++)
    cout << "dy(n/2)/dy0[" << i << "]=" << y_a[i] << endl;
  return 0;
}
```

B PDE / Implicit Scheme

Listing 24: Primal PDE / Implicit Scheme

```
#ifndef __F_H_INCLUDED_
#define __F_H_INCLUDED_

#include <Eigen/LU>
using namespace Eigen;
```

```
6
7  // rhs of ode
8  template <typename T, int N=Dynamic>
9  inline void g(const Matrix<T,3,1>& p, const Matrix<T,N,1>& y,
10                Matrix<T,N,1>& r) {
11   int n=y.size();
12   for (int i=0;i<n;i++) {
13     r(i)=p(0)*(n+1)*(n+1);
14     if (i==0)
15       r(i)*=p(1)-2*y(i)+y(i+1);
16     else if (i==n-1)
17       r(i)*=y(i-1)-2*y(i)+p(2);
18     else
19       r(i)*=y(i-1)-2*y(i)+y(i+1);
20   }
21 }
22
23 // tangent of rhs of ode
24 template <typename T, int N=Dynamic>
25 inline void g_t(const Matrix<T,3,1>& p, const Matrix<T,N,1>& y,
26                 const Matrix<T,N,1>& y_t, Matrix<T,N,1>& r_t) {
27   int n=y.size();
28   for (int i=0;i<n;i++) {
29       r_t(i)=p(0)*(n+1)*(n+1);
30     if (i==0)
31       r_t(i)*=-2*y_t(i)+y_t(i+1);
32     else if (i==n-1)
33       r_t(i)*=y_t(i-1)-2*y_t(i);
34     else
35       r_t(i)*=y_t(i-1)-2*y_t(i)+y_t(i+1);
36   }
37 }
38
39 // Jacobian of rhs of ode
40 template <typename T, int N=Dynamic>
41 inline void dgdy(const Matrix<T,3,1>& p,
42                  const Matrix<T,N,1>& y, Matrix<T,N,N>& A) {
43   int n=y.size();
44   Matrix<T,N,1> r=Matrix<T,N,1>::Zero(n), r_t=Matrix<T,N,1>::Zero(n);
45   for (int i=0;i<n;i++) {
46     Matrix<T,N,1> y_t=Matrix<T,N,1>::Zero(n);
47     y_t(i)=1;
48     g_t(p,y,y_t,r_t);
49     if (i>0) A(i,i-1)=r_t(i-1);
50     A(i,i)=r_t(i);
51     if (i<n-1) A(i,i+1)=r_t(i+1);
```

```
52    }
53  }
54
55  template <typename T, int N=Dynamic>
56  inline void dfdy(const int m, const Matrix<T,3,1>& p,
57                   const Matrix<T,N,1>& y, Matrix<T,N,N>& A) {
58    int n=y.size();
59    dgdy(p,y,A);
60    A=Matrix<T,N,N>::Identity(n,n)-A/m;
61  }
62
63  // residual of nls
64  template <typename T, int N=Dynamic>
65  inline void f(const int m, const Matrix<T,3,1>& p,
66                const Matrix<T,N,1>& y, const Matrix<T,N,1>& y_prev,
67                Matrix<T,N,1>& r) {
68    g(p,y,r); r=y-y_prev-r/m;
69  }
70
71  // Newton solver for nls
72  template <typename T, int N=Dynamic>
73  inline void newton(const int m, const Matrix<T,3,1>& p,
74                     const Matrix<T,N,1>& y_prev, Matrix<T,N,1>& y) {
75    int n=y.size();
76    const double eps=1e-12;
77    Matrix<T,N,N> A=Matrix<T,N,N>::Zero(n,n);
78    Matrix<T,N,1> r=Matrix<T,N,1>::Zero(n);
79    f(m,p,y,y_prev,r);
80    while (r.norm()>eps) {
81      dfdy(m,p,y,A);
82      PartialPivLU<Matrix<T,N,N>> LU(A);
83      y-=LU.solve(r);
84      f(m,p,y,y_prev,r);
85    }
86  }
87
88  // implicit Euler integration
89  template <typename T, int N=Dynamic>
90  inline void f(const int m, const Matrix<T,3,1>& p, Matrix<T,N,1>& y) {
91    for (int j=0;j<m;j++) {
92      Matrix<T,N,1> y_prev=y;
93      newton(m,p,y_prev,y);
94    }
95  }
96
97  #endif
```

Listing 25: Primal PDE / Implicit Scheme (Driver)

```
#include "std_includes.h"
#include "f.h"

int main(int c, char* v[]){
  assert(c==3);
  int n=atoi(v[1]), m=atoi(v[2]);
  Matrix<double,Dynamic,1> y(n);
  for (int i=0;i<n;i++) y(i)=(i+1)*log(static_cast<double>(i+2));
  Matrix<double,3,1> p(3); p(0)=1e-4; p(1)=42; p(2)=0;
  f(m,p,y);
  cout << 0 << " " << p(1) << endl;
  for (int i=0;i<n;i++)
    cout << static_cast<double>(i+1)/(n+1) << " " << y(i) << endl;
  cout << 1 << " " << p(2) << endl;
  return 0;
}
```

B.1 Tangents

Listing 26: Symbolic Tangent PDE / Implicit Scheme (Handwritten)

```
#include "std_includes.h"
#include "f.h"

template <typename T, int N=Dynamic>
inline void step_t(const int m, const Matrix<T,3,1>& p, Matrix<T,N,1>& y,
                   Matrix<T,N,1>& y_t) {
  int n=y.size();
  Matrix<T,N,N> A=Matrix<T,N,N>::Zero(n,n);
  Matrix<T,N,1> y_prev=y;
  newton(m,p,y_prev,y);
  dfdy(m,p,y,A);
  PartialPivLU<Matrix<T,N,N>> LU(A);
  y_t=LU.solve(y_t);
}

template <typename T, int N=Dynamic>
inline void f_t(const int m, const Matrix<T,3,1>& p, Matrix<T,N,1>& y,
                Matrix<T,N,1>& y_t) {
  for (int j=0;j<m;j++) step_t(m,p,y,y_t);
}

int main(int c, char* v[]){
  assert(c==3);
  int n=atoi(v[1]), m=atoi(v[2]);
```

```
25   Matrix<double,Dynamic,1> y=Matrix<double,Dynamic,1>::Zero(n);
26   Matrix<double,3,1> p=Matrix<double,3,1>::Zero(3);
27   p(0)=1e-3; p(1)=42; p(2)=0;
28   for (int j=0;j<n;j++) {
29     for (int i=0;i<n;i++) y(i)=(i+1)*log(static_cast<double>(i+2));
30     Matrix<double,Dynamic,1> y_t=Matrix<double,Dynamic,1>::Zero(n);
31     y_t(j)=1;
32     f_t(m,p,y,y_t);
33     cout << "dy(n/2)/dy0[" << j << "]=" << y_t(n/2) << endl;
34   }
35   return 0;
36 }
```

Listing 27: Algorithmic Tangent PDE / Implicit Scheme (dco/c++)

```
1  #include "std_includes.h"
2
3  #include "dco.hpp"
4  typedef dco::gt1s<double>::type DCO_T;
5
6  #include "f.h"
7
8  int main(int c, char* v[]){
9    assert(c==3);
10   int n=atoi(v[1]), m=atoi(v[2]);
11   Matrix<DCO_T,Dynamic,1> y(n);
12   Matrix<DCO_T,3,1> p; p(0)=1e-3; p(1)=42; p(2)=0;
13   for (int j=0;j<n;j++) {
14     for (int i=0;i<n;i++) y(i)=(i+1)*log(static_cast<double>(i+2));
15     dco::derivative(y(j))=1;
16     f(m,p,y);
17     cout << "dy(n/2)/dy0[" << j << "]=" << dco::derivative(y(n/2)) << endl;
18   }
19   return 0;
20 }
```

B.2 Adjoints

Listing 28: Algorithmic Adjoint PDE / Implicit Scheme (dco/c++)

```
1  #include "std_includes.h"
2
3  #include "dco.hpp"
4  typedef dco::ga1sm<double> DCO_AM;
5  typedef DCO_AM::type DCO_A;
6  typedef DCO_AM::tape_t DCO_AM_TAPE;
7
```

```
#include "f.h"

int main(int c, char* v[]){
  assert(c==3);
  int n=atoi(v[1]), m=atoi(v[2]);
  Matrix<DCO_A,Dynamic,1> y0=Matrix<DCO_A,Dynamic,1>::Zero(n);
  for (int i=0;i<n;i++) y0(i)=(i+1)*log(static_cast<double>(i+2));
  Matrix<DCO_A,3,1> p=Matrix<DCO_A,3,1>::Zero(3);
  p(0)=1e-3; p(1)=42; p(2)=0;
  DCO_AM_TAPE* tape=DCO_AM_TAPE::create();
  for(int i=0;i<n;i++) tape->register_variable(y0(i));
  Matrix<DCO_A,Dynamic,1> y=y0;
  f(m,p,y);
  for(int i=0;i<n;i++) tape->register_output_variable(y(i));
  dco::derivative(y(n/2))=1.;
  tape->interpret_adjoint();
  for(int i=0;i<n;i++)
    cout << "dy(n/2)/dy0[" << i << "]=" << dco::derivative(y0(i)) << endl;
  DCO_AM_TAPE::remove(tape);
  return 0;
}
```

Listing 29: Algorithmic Adjoint PDE with Symbolic Adjoint Nonlinear Euler System (Handwritten)

```
#include "std_includes.h"
#include "f.h"

enum Mode { AUGMENTED_PRIMAL, SPLIT_ADJOINT };

template <typename T, int N=Dynamic>
inline void step_a(Mode mode, const int m, const Matrix<T,3,1>& p,
                   Matrix<T,N,1>& y, Matrix<T,N,1>& y_a) {
  static stack<Matrix<T,N,1>> psols;
  int n=y.size();
  Matrix<T,N,N> A=Matrix<T,N,N>::Zero(n,n);
  switch (mode) {
  case AUGMENTED_PRIMAL: {
    Matrix<T,N,1> y_prev=Matrix<T,N,1>::Zero(n);
    y_prev=y;
    newton(m,p,y_prev,y);
    psols.push(y);
    break;
  }
  case SPLIT_ADJOINT: {
    y=psols.top(); psols.pop();
    dfdy(m,p,y,A);
```

```
23      PartialPivLU<Matrix<T,N,N>> LU(A.transpose());
24      y_a=LU.solve(y_a);
25      break;
26    }
27    }
28  }
29
30  template <typename T, int N=Dynamic>
31  inline void f_a(const int m, const Matrix<T,3,1>& p, Matrix<T,N,1>& y,
32                  Matrix<T,N,1>& y_a) {
33    for (int j=0;j<m;j++)
34      step_a(AUGMENTED_PRIMAL,m,p,y,y_a);
35    for (int j=0;j<m;j++)
36      step_a(SPLIT_ADJOINT,m,p,y,y_a);
37  }
38
39  int main(int c, char* v[]){
40    assert(c==3);
41    int n=atoi(v[1]), m=atoi(v[2]);
42    Matrix<double,Dynamic,1> y(n);
43    for (int i=0;i<n;i++) y(i)=(i+1)*log(static_cast<double>(i+2));
44    Matrix<double,3,1> p; p(0)=1e-3; p(1)=42; p(2)=0;
45    Matrix<double,Dynamic,1> y_a=Matrix<double,Dynamic,1>::Zero(n);
46    y_a(n/2)=1;
47    f_a(m,p,y,y_a);
48    for(int i=0;i<n;i++)
49      cout << "dy(n/2)/dy0[" << i << "]=" << y_a(i) << endl;
50    return 0;
51  }
```

Listing 30: Symbolic Adjoint PDE / Implicit Scheme with Equidistant Checkpointing (Handwritten)

```
1  #include "std_includes.h"
2  #include "f.h"
3
4  enum Mode { PRIMAL, AUGMENTED_PRIMAL, SPLIT_ADJOINT };
5
6  template <typename T, int N=Dynamic>
7  inline void step_a(Mode mode, const int m, const Matrix<T,3,1>& p,
8                     Matrix<T,N,1>& y, Matrix<T,N,1>& y_a) {
9    static stack<Matrix<T,N,1>> psols;
10   int n=y.size();
11   Matrix<T,N,N> A=Matrix<T,N,N>::Zero(n,n);
12   switch (mode) {
13   case PRIMAL: {
14     Matrix<T,N,1> y_prev=y;
```

```
15      newton(m,p,y_prev,y);
16      break;
17    }
18    case AUGMENTED_PRIMAL: {
19      Matrix<T,N,1> y_prev=y;
20      newton(m,p,y_prev,y);
21      psols.push(y);
22      break;
23    }
24    case SPLIT_ADJOINT: {
25      y=psols.top(); psols.pop();
26      dfdy(m,p,y,A);
27      PartialPivLU<Matrix<T,N,N>> LU(A.transpose());
28      y_a=-LU.solve(-y_a);
29      break;
30    }
31    }
32  }
33
34  template <typename T, int N=Dynamic>
35  inline void f_a(const int m, const int ncs, const Matrix<T,3,1>& p,
36                  Matrix<T,N,1>& y, Matrix<T,N,1>& y_a) {
37    static stack<Matrix<T,N,1>> cp;
38    for (int j=0;j<m-ncs;j+=ncs) {
39      cp.push(y);
40      for (int i=0;i<ncs;i++)
41        step_a(PRIMAL,m,p,y,y_a);
42    }
43    for (int i=0;i<ncs;i++)
44      step_a(AUGMENTED_PRIMAL,m,p,y,y_a);
45    for (int i=0;i<ncs;i++)
46      step_a(SPLIT_ADJOINT,m,p,y,y_a);
47    for (int j=0;j<m-ncs;j+=ncs) {
48      y=cp.top(); cp.pop();
49      for (int i=0;i<ncs;i++)
50        step_a(AUGMENTED_PRIMAL,m,p,y,y_a);
51      for (int i=0;i<ncs;i++)
52        step_a(SPLIT_ADJOINT,m,p,y,y_a);
53    }
54  }
55
56  int main(int c, char* v[]){
57    assert(c==4);
58    int n=atoi(v[1]), m=atoi(v[2]), ncs=atoi(v[3]); assert(m%ncs==0);
59    Matrix<double,Dynamic,1> y(n);
60    for (int i=0;i<n;i++) y(i)=(i+1)*log(static_cast<double>(i+2));
```

```
  Matrix<double,3,1> p; p(0)=1e-3; p(1)=42; p(2)=0;
  Matrix<double,Dynamic,1> y_a=Matrix<double,Dynamic,1>::Zero(n);
  y_a(n/2)=1;
  f_a(m,ncs,p,y,y_a);
  for(int i=0;i<n;i++)
    cout << "dy(n/2)/dy0[" << i << "]=" << y_a(i) << endl;
  return 0;
}
```

Listing 31: Symbolic Adjoint PDE / Explicit Scheme (Handwritten)

```
#include "std_includes.h"
#include "f.h"

enum Mode { AUGMENTED_PRIMAL, SPLIT_ADJOINT };

template <typename T, int N=Dynamic>
inline void step_a(Mode mode, const int m, const Matrix<T,3,1>& p,
                   Matrix<T,N,1>& y, Matrix<T,N,1>& y_a) {
  static stack<Matrix<T,N,1>> psols;
  int n=y.size();
  Matrix<T,N,N> A=Matrix<T,N,N>::Zero(n,n);
  switch (mode) {
  case AUGMENTED_PRIMAL: {
    Matrix<T,N,1> y_prev=Matrix<T,N,1>::Zero(n);
    y_prev=y;
    newton(m,p,y_prev,y);
    psols.push(y);
    break;
  }
  case SPLIT_ADJOINT: {
    y=psols.top(); psols.pop();
    dgdy(p,y,A);
    y_a=y_a+A.transpose()*y_a/m;
    break;
  }
  }
}

template <typename T, int N=Dynamic>
inline void f_a(const int m, const Matrix<T,3,1>& p, Matrix<T,N,1>& y,
                Matrix<T,N,1>& y_a) {
  for (int j=0;j<m;j++)
    step_a(AUGMENTED_PRIMAL,m,p,y,y_a);
  for (int j=0;j<m;j++)
    step_a(SPLIT_ADJOINT,m,p,y,y_a);
}
```

```
37
38 int main(int c, char* v[]){
39   assert(c==3);
40   int n=atoi(v[1]), m=atoi(v[2]);
41   Matrix<double,Dynamic,1> y(n);
42   for (int i=0;i<n;i++) y(i)=(i+1)*log(static_cast<double>(i+2));
43   Matrix<double,3,1> p; p(0)=1e-3; p(1)=42; p(2)=0;
44   Matrix<double,Dynamic,1> y_a=Matrix<double,Dynamic,1>::Zero(n);
45   y_a(n/2)=1;
46   f_a(m,p,y,y_a);
47   for(int i=0;i<n;i++)
48     cout << "dy(n/2)/dy0[" << i << "]=" << y_a(i) << endl;
49   return 0;
50 }
```

C LIBOR

Listing 32: Primal LIBOR

```
1  #include "std_includes.h"
2
3  const int p=10;
4  const int m=40;
5  const int n=m+40;
6  const int no=15;
7
8  const double delta=0.25;
9  const vector<int> maturities({4,4,4,8,8,8,20,20,20,28,28,28,40,40,40});
10 const vector<double> swaprates({.045,.05,.055,.045,.05,.055,.045,.05,
11                                 .055,.045,.05,.055,.045,.05,.055});
12 const vector<double> sigma(n,0.2);
13
14 template <typename T>
15 inline void path_calc(
16     const int path,
17     vector<T>& L,
18     const vector<vector<double>>& Z
19 ) {
20   for(int j=0;j<m;j++) {
21     double aux1=sqrt(delta)*Z[path][j];
22     T S=0.0;
23     for (int i=j+1;i<n;i++) {
24       double aux2=delta*sigma[i-j-1];
25       S+=(aux2*L[i])/(1.0+delta*L[i]);
26       L[i]=L[i]*exp(aux2*S+sigma[i-j-1]*(aux1-0.5*aux2));
27     }
```

```
28    }
29  }
30
31  template <typename T>
32  inline void portfolio( const vector<T>& L, T& P ) {
33    vector<T> B(n),S(n);
34    T b=1.0;
35    T s=0.0;
36    for (int j=m;j<n;j++) {
37      b=b/(1.0+delta*L[j]); B[j]=b;
38      s=s+delta*b; S[j]=s;
39    }
40    P=0;
41    for (int i=0;i<no;i++){
42      int j=maturities[i]+m-1;
43      T swapval=B[j]+swaprates[i]*S[j]-1.0;
44      if (swapval<0) P+=-100.0*swapval;
45    }
46    for (int i=0;i<m;i++) P=P/(1.0+delta*L[i]);
47  }
48
49  template<typename T>
50  inline void f(const vector<T>& L, T& P, const vector<vector<double>>& Z) {
51    T Ps=0;
52    for (int j=0;j<p;j++) {
53      vector<T> Lc(L);
54      path_calc(j,Lc,Z);
55      portfolio(Lc,P);
56      Ps+=P;
57    }
58    P=Ps/p;
59  }
```

Listing 33: Primal LIBOR (Driver)

```
1   #include "std_includes.h"
2   #include "f.h"
3
4   int main() {
5     vector<double> L(n,0.05); double P=0;
6     srand(0); default_random_engine generator(0);
7     normal_distribution<double> distribution(0.0,1.0);
8     vector<vector<double>> Z(p,vector<double>(m));
9     for (int j=0; j<p;j++)
10      for (int i=0;i<m;i++)
11        Z[j][i]=0.3+distribution(generator);
12    f(L,P,Z);
```

```
13    cout << "P=" << P << endl;
14    return 0;
15  }
```

C.1 First-Order AD

C.1.1 Tangents

Listing 34: Tangent LIBOR (Handwritten)

```
1   #include "std_includes.h"
2   #include "f.h"
3
4   template <typename T>
5   inline void path_calc_t(
6       const int path,
7       vector<T>& L,
8       vector<T>& L_t,
9       const vector<vector<double>>& Z
10  ) {
11    for(int j=0;j<m;j++) {
12      double aux1=sqrt(delta)*Z[path][j];
13      T S_t=0.0; T S=0.0;
14      for (int i=j+1;i<n;i++) {
15        double aux2=delta*sigma[i-j-1];
16        S_t+=(aux2/(1+delta*L[i])-delta*aux2*L[i]
17             /((1+delta*L[i])*(1+delta*L[i])))*L_t[i];
18        S+=(aux2*L[i])/(1.0+delta*L[i]);
19        // L[i]=L[i]*exp(aux2*S+sigma[i-j-1]*(aux1-0.5*aux2));
20        T t1_t=aux2*exp(aux2*S+sigma[i-j-1]*(aux1-0.5*aux2))*S_t;
21        T t1=exp(aux2*S+sigma[i-j-1]*(aux1-0.5*aux2));
22        L_t[i]=L_t[i]*t1+L[i]*t1_t;
23        L[i]=L[i]*t1;
24      }
25    }
26  }
27
28  template <typename T>
29  inline void portfolio_t(
30      const vector<T>& L, const vector<T>& L_t, T& P, T& P_t) {
31    vector<T> B(n),B_t(n),S(n),S_t(n);
32    T b_t=0.0; T b=1.0;
33    T s_t=0.0; T s=0.0;
34    for (int j=m;j<n;j++) {
35      b_t=b_t/(1.0+delta*L[j])-delta*b*L_t[j]
36          /((1.0+delta*L[j])*(1.0+delta*L[j]));
37      b=b/(1.0+delta*L[j]);
```

```
38      B_t[j]=b_t;
39      B[j]=b;
40      s_t=s_t+delta*b_t;
41      s=s+delta*b;
42      S_t[j]=s_t;
43      S[j]=s;
44    }
45    P_t=0; P=0;
46    for (int i=0;i<no;i++){
47      int j=maturities[i]+m-1;
48      T swapval_t=B_t[j]+swaprates[i]*S_t[j];
49      T swapval=B[j]+swaprates[i]*S[j]-1.0;
50      if (swapval<0) {
51        P_t+=-100.0*swapval_t;
52        P+=-100.0*swapval;
53      }
54    }
55    for (int i=0;i<m;i++) {
56      P_t=P_t/(1.0+delta*L[i])-delta*P*L_t[i]
57          /((1.0+delta*L[i])*(1.0+delta*L[i]));
58      P=P/(1.0+delta*L[i]);
59    }
60  }
61
62  template<typename T>
63  inline void f(const vector<T>& L, const vector<T>& L_t,
64      T& P, T& P_t, const vector<vector<double>>& Z) {
65    T Ps_t=0;
66    T Ps=0;
67    for (int j=0;j<p;j++) {
68      vector<T> Lc_t(L_t);
69      vector<T> Lc(L);
70      path_calc_t(j,Lc,Lc_t,Z);
71      portfolio_t(Lc,Lc_t,P,P_t);
72      Ps_t+=P_t; Ps+=P;
73    }
74    P_t=Ps_t/p; P=Ps/p;
75  }
76
77
78  int main() {
79    vector<double> L(n,0.05); double P=0;
80    vector<double> L_t(n,0); double P_t=0;
81    srand(0); default_random_engine generator(0);
82    normal_distribution<double> distribution(0.0,1.0);
83    vector<vector<double>> Z(p,vector<double>(m));
```

```
84    for (int k=0;k<n;k++) {
85      generator.seed(0);
86      for (int j=0;j<p;j++)
87        for (int i=0;i<m;i++)
88          Z[j][i]=0.3+distribution(generator);
89      for (int i=0;i<n;i++) { L[i]=0.05; L_t[i]=0.0; }
90      L_t[k]=1.0;
91      f(L,L_t,P,P_t,Z);
92      cout << "dPdL[" << k << "]=" << P_t << endl;
93    }
94    return 0;
95  }
```

Listing 35: Tangent LIBOR (dco/c++)

```
1   #include "std_includes.h"
2   #include "f.h"
3
4   #include "dco.hpp"
5   typedef dco::gt1s<double>::type DCO_T;
6
7   int main() {
8     vector<DCO_T> L(n,0.05); DCO_T P=0;
9     srand(0); default_random_engine generator(0);
10    normal_distribution<double> distribution(0.0,1.0);
11    vector<vector<double>> Z(p,vector<double>(m));
12    for (int j=0; j<p;j++)
13      for (int i=0;i<m;i++)
14        Z[j][i]=0.3+distribution(generator);
15    vector<double> dPdL(n,0);
16    for (int i=0;i<n;i++) {
17      dco::derivative(L[i])=1;
18      f(L,P,Z);
19      dco::derivative(L[i])=0;
20      dPdL[i]=dco::derivative(P);
21    }
22    for (int i=0;i<n;i++)
23      cout << "dPdL[" << i << "]=" << dPdL[i] << endl;
24    return 0;
25  }
```

Listing 36: Vector Tangent LIBOR (dco/c++)

```
1   #include "std_includes.h"
2   #include "f.h"
3
4   #include "dco.hpp"
```

```
typedef dco::gt1v<double,n>::type DCO_T;

int main() {
  vector<DCO_T> L(n,0.05); DCO_T P=0;
  srand(0); default_random_engine generator(0);
  normal_distribution<double> distribution(0.0,1.0);
  vector<vector<double>> Z(p,vector<double>(m));
  for (int j=0; j<p;j++)
    for (int i=0;i<m;i++)
      Z[j][i]=0.3+distribution(generator);
  for (int i=0;i<n;i++) dco::derivative(L[i])[i]=1;
  f(L,P,Z);
  vector<double> dPdL(n,0);
  for (int i=0;i<n;i++) dPdL[i]=dco::derivative(P)[i];
  for (int i=0;i<n;i++)
    cout << "dPdL[" << i << "]=" << dPdL[i] << endl;
  return 0;
}
```

C.1.2 Adjoints

Listing 37: Adjoint LIBOR (Handwritten)

```
#include "f.h"

enum Mode { AUGMENTED_PRIMAL, SPLIT_ADJOINT };

stack<double> tbr_d;
stack<int> tbr_i;
stack<int> tbr_f;

template <typename T>
void a_path_calc(
    Mode mode,
    vector<T>& L,
    vector<T>& a_L,
    const vector<double>& Z
) {
  double aux1=0,aux2=0;
  T S=0, a_S=0;
  switch (mode) {
    case AUGMENTED_PRIMAL:
      for(int j=0;j<m;j++) {
        tbr_d.push(aux1);
        aux1=sqrt(delta)*Z[j];
        tbr_d.push(S);
        S=0;
```

```
25          for (int i=j+1;i<n;i++) {
26            tbr_d.push(aux2);
27            aux2=delta*sigma[i-j-1];
28            tbr_d.push(S);
29            S+=(aux2*L[i])/(1.0+delta*L[i]);
30            tbr_d.push(L[i]);
31            L[i]=L[i]*exp(aux2*S+sigma[i-j-1]*(aux1-0.5*aux2));
32          }
33        }
34        tbr_d.push(aux1);
35        tbr_d.push(aux2);
36        tbr_d.push(S);
37        break;
38      case SPLIT_ADJOINT:
39        a_S=0;
40        S=tbr_d.top(); tbr_d.pop();
41        aux2=tbr_d.top(); tbr_d.pop();
42        aux1=tbr_d.top(); tbr_d.pop();
43        for(int j=m-1;j>=0;j--) {
44          for (int i=n-1;i>=j+1;i--) {
45            L[i]=tbr_d.top(); tbr_d.pop();
46            a_S+=aux2*L[i]*exp(aux2*S+sigma[i-j-1]*(aux1-0.5*aux2))*a_L[i];
47            a_L[i]=exp(aux2*S+sigma[i-j-1]*(aux1-0.5*aux2))*a_L[i];
48            S=tbr_d.top(); tbr_d.pop();
49            a_L[i]+=(aux2/(1+L[i]*delta)-delta*aux2*L[i]
50                    /((1+L[i]*delta)*(1+L[i]*delta)))*a_S;
51            aux2=tbr_d.top(); tbr_d.pop();
52          }
53          a_S=0;
54          S=tbr_d.top(); tbr_d.pop();
55          aux1=tbr_d.top(); tbr_d.pop();
56        }
57        break;
58    }
59  }
60
61  template <typename T>
62  void a_portfolio(
63      Mode mode,
64      const vector<T>& L,
65      vector<T>& a_L,
66      T& P,
67      T& a_P
68  ) {
69    vector<T> B(n,0),S(n,0);
70    T swapval=0,b=0,s=0;
```

```
71    T a_swapval=0,a_b=0,a_s=0;
72    vector<T> a_B(n,0),a_S(n,0);
73    switch (mode) {
74      case AUGMENTED_PRIMAL:
75        b=1.0;
76        s=0.0;
77        for (int j=m;j<n;j++) {
78          tbr_d.push(b);
79          b=b/(1.0+delta*L[j]);
80          B[j]=b;
81          s=s+delta*b;
82          S[j]=s;
83        }
84        P=0;
85        for (int i=0;i<no;i++){
86          int j=maturities[i]+m-1;
87          tbr_i.push(j);
88          swapval=B[j]+swaprates[i]*S[j]-1.0;
89          if (swapval<0) {
90            P+=-100.0*swapval;
91            tbr_f.push(1);
92          } else tbr_f.push(0);
93        }
94        for (int i=0;i<m;i++) {
95          tbr_d.push(P);
96          P=P/(1.0+delta*L[i]);
97        }
98        tbr_d.push(b);
99        break;
100     case SPLIT_ADJOINT:
101       b=tbr_d.top(); tbr_d.pop();
102       for (int i=m-1;i>=0;i--) {
103         P=tbr_d.top(); tbr_d.pop();
104         a_L[i]+=-P*a_P*delta/((1.0+delta*L[i])*(1.0+delta*L[i]));
105         a_P=a_P/(1.0+delta*L[i]);
106       }
107       for (int i=no-1;i>=0;i--) {
108         if (tbr_f.top()) a_swapval+=-100.0*a_P;
109         tbr_f.pop();
110         int j=tbr_i.top(); tbr_i.pop();
111         a_B[j]+=a_swapval;
112         a_S[j]+=swaprates[i]*a_swapval; a_swapval=0.0;
113       }
114       a_P=0.0;
115       for (int j=n-1;j>=m;j--) {
116         a_s+=a_S[j]; a_S[j]=0;
```

```
        a_b+=delta*a_s;
        a_b+=a_B[j]; a_B[j]=0;
        b=tbr_d.top(); tbr_d.pop();
        a_L[j]+=-delta*b*a_b/((1.0+delta*L[j])*(1.0+delta*L[j]));
        a_b=a_b/(1.0+delta*L[j]);
      }
      a_b=0.0;
      a_s=0.0;
      break;
  }
}

int main() {
  vector<double> Z(n,0),L(n,0),a_L(n,0),Li(n,0.05),a_Li(n,0);
  double P=0,a_P=0;
  default_random_engine generator(0);
  normal_distribution<double> distribution(0.0,1.0);

  for (int j=0;j<p;j++) {
    for (int i=0;i<m;i++)
      Z[i]=0.3+distribution(generator);
    for (int i=0;i<n;i++) L[i]=Li[i];
    a_path_calc(AUGMENTED_PRIMAL,L,a_L,Z);
    a_portfolio(AUGMENTED_PRIMAL,L,a_L,P,a_P);
    a_P=1.0/p;
    a_portfolio(SPLIT_ADJOINT,L,a_L,P,a_P);
    a_path_calc(SPLIT_ADJOINT,L,a_L,Z);
    for (int i=0;i<n;i++) { a_Li[i]+=a_L[i]; a_L[i]=0; }
  }
  for (int i=0;i<n;i++)
    cout << "dPdL[" << i << "]=" << a_Li[i] << endl;
  return 0;
}
```

Listing 38: Adjoint LIBOR (dco/c++)

```
#include "std_includes.h"
#include "f.h"

#include "dco.hpp"
typedef dco::ga1s<double> DCO_AM;
typedef typename DCO_AM::type DCO_A;
typedef typename DCO_AM::tape_t DCO_AM_TAPE;

int main() {
  vector<DCO_A> L(n,0.05); DCO_A P=0;
  srand(0); default_random_engine generator(0);
```

```
12   normal_distribution<double> distribution(0.0,1.0);
13   vector<vector<double>> Z(p,vector<double>(m));
14   for (int j=0; j<p;j++)
15     for (int i=0;i<m;i++)
16       Z[j][i]=0.3+distribution(generator);
17   DCO_AM::global_tape=DCO_AM_TAPE::create();
18   DCO_AM::global_tape->register_variable(L);
19   f(L,P,Z);
20   DCO_AM::global_tape->register_output_variable(P);
21   dco::derivative(P)=1;
22   DCO_AM::global_tape->interpret_adjoint();
23   vector<double> dPdL(dco::derivative(L));
24   cerr << dco::size_of(DCO_AM::global_tape) << "B" << endl;
25   DCO_AM_TAPE::remove(DCO_AM::global_tape);
26   for(int i=0;i<n;i++)
27     cout << "dPdL[" << i << "]=" << dPdL[i] << endl;
28   return 0;
29 }
```

C.2 Second-Order AD

C.2.1 Tangents

Listing 39: Second-Order Tangent LIBOR (dco/c++)

```
1  #include "std_includes.h"
2  #include "f.h"
3
4  #include "dco.hpp"
5  typedef dco::gt1s<double>::type DCO_T;
6  typedef dco::gt1s<DCO_T>::type DCO_TT;
7
8  int main() {
9    vector<DCO_TT> L(n,0.05); DCO_TT P=0;
10   srand(0); default_random_engine generator(0);
11   normal_distribution<double> distribution(0.0,1.0);
12   vector<vector<double>> Z(p,vector<double>(m));
13   for (int j=0; j<p;j++)
14     for (int i=0;i<m;i++)
15       Z[j][i]=0.3+distribution(generator);
16   vector<vector<double> > ddPdLL(n,vector<double>(n,0));
17   for (int i=0;i<n;i++) {
18     dco::value(dco::derivative(L[i]))=1;
19     for (int j=0;j<=i;j++) {
20       dco::derivative(dco::value(L[j]))=1;
21       f(L,P,Z);
22       ddPdLL[i][j]=ddPdLL[j][i]=dco::derivative(dco::derivative(P));
```

```
23        dco::derivative(dco::value(L[j]))=0;
24      }
25      dco::value(dco::derivative(L[i]))=0;
26    }
27    for (int i=0;i<n;i++)
28      for (int j=0;j<n;j++)
29        cout << "ddP/dL[" << i << "]dL[" << j << "]=" << ddPdLL[i][j] << endl;
30    return 0;
31  }
```

C.2.2 Adjoints

Listing 40: Second-Order Adjoint LIBOR (dco/c++)

```
1   #include "std_includes.h"
2   #include "f.h"
3
4   #include "dco.hpp"
5   typedef dco::gt1s<double>::type DCO_T;
6   typedef dco::ga1s<DCO_T> DCO_TAM;
7   typedef DCO_TAM::type DCO_TA;
8   typedef DCO_TAM::tape_t DCO_TAM_TAPE;
9
10  int main() {
11    vector<DCO_TA> L(n,0.05); DCO_TA P=0;
12    srand(0); default_random_engine generator(0);
13    normal_distribution<double> distribution(0.0,1.0);
14    vector<vector<double>> Z(p,vector<double>(m));
15    for (int j=0; j<p;j++)
16      for (int i=0;i<m;i++)
17        Z[j][i]=0.3+distribution(generator);
18    DCO_TAM::global_tape=DCO_TAM_TAPE::create();
19    DCO_TAM::global_tape->register_variable(L);
20    DCO_TAM_TAPE::position_t tpos=DCO_TAM::global_tape->get_position();
21    vector<vector<double> > ddPdLL(n,vector<double>(n,0));
22    for(int j=0;j<n;j++) {
23      dco::derivative(dco::value(L[j]))=1;
24      f(L,P,Z);
25      DCO_TAM::global_tape->register_output_variable(P);
26      dco::value(dco::derivative(P))=1;
27      DCO_TAM::global_tape->interpret_adjoint_to(tpos);
28      for(int i=0;i<n;i++) {
29        ddPdLL[i][j]=dco::derivative(dco::derivative(L[i]));
30        dco::derivative(L[i])=0;
31      }
32      dco::derivative(dco::value(L[j]))=0;
33      DCO_TAM::global_tape->reset_to(tpos);
```

```
  }
  DCO_TAM_TAPE::remove(DCO_TAM::global_tape);
  for (int i=0;i<n;i++)
    for (int j=0;j<n;j++)
      cout << "ddP/dL[" << i << "]dL[" << j << "]=" << ddPdLL[i][j] << endl;
  return 0;
}
```

D Product Reduction

D.1 First-Order AD

D.1.1 Tangents

Listing 41: Tangent Product Reduction (Handwritten)

```
#include "std_includes.h"
using namespace std;

template<typename T>
void f_t(const vector<T>& x, const vector<T>& x_t, T& y, T& y_t) {
  assert(x.size()>0);
  y_t=x_t[0]; y=x[0];
  for (size_t i=1;i<x.size();i++) { y_t=y_t*x[i]+y*x_t[i]; y*=x[i]; }
}

void driver(vector<double>& x, double &y, vector<double>& g) {
  vector<double> x_t(x.size(),0);
  for (size_t i=0;i<x.size();i++) {
    x_t[i]=1;
    f_t(x,x_t,y,g[i]);
    x_t[i]=0;
  }
}

int main(int c, char* v[]) {
  assert(c==2); int n=atoi(v[1]); assert(n>0);
  vector<double> x(n), g(n); double y;
  for (int i=0;i<n;i++) x[i]=cos(static_cast<double>(i));
  driver(x,y,g);
  cout << y << endl;
  for (int i=0;i<n;i++) cout << g[i] << endl;
  return 0;
}
```

D.1.2 Adjoints

Listing 42: Adjoint Product Reduction (Handwritten)

```
#include "std_includes.h"
using namespace std;

template<typename T>
void f_a(const vector<T>& x, vector<T>& x_a, T& y, T& y_a) {
  assert(x.size()>0);
  stack<T> tbr;
  y=x[0];
  for (size_t i=1;i<x.size();i++) { tbr.push(y); y*=x[i]; }
  double ys=y;
  for (size_t i=x.size()-1;i>0;i--) {
    y=tbr.top(); tbr.pop(); x_a[i]+=y*y_a; y_a=x[i]*y_a;
  }
  x_a[0]=y_a; y_a=0;
  y=ys;
}

void driver(vector<double>& x, double &y, vector<double>& g) {
  double y_a=1;
  f_a(x,g,y,y_a);
}

int main(int c, char* v[]) {
  assert(c==2); int n=atoi(v[1]); assert(n>0);
  vector<double> x(n), g(n); double y;
  for (int i=0;i<n;i++) x[i]=cos(static_cast<double>(i));
  driver(x,y,g);
  cout << y << endl;
  for (int i=0;i<n;i++) cout << g[i] << endl;
  return 0;
}
```

D.2 Second-Order AD

D.2.1 Tangents

Listing 43: Second-Order Tangent Product Reduction (Handwritten)

```
#include "std_includes.h"
using namespace std;

template<typename T>
void f_tt(
    const vector<T>& x,
    const vector<T>& x_t,
    const vector<T>& xt,
```

```
9       const vector<T>& xt_t,
10      T& y,
11      T& y_t,
12      T& yt,
13      T& yt_t
14  ) {
15    assert(x.size()>0);
16    yt_t=xt_t[0];
17    yt=xt[0];
18    y_t=x_t[0];
19    y=x[0];
20    for (size_t i=1;i<x.size();i++) {
21      yt_t=yt_t*x[i]+yt*x_t[i]+y_t*xt[i]+y*xt_t[i];
22      yt=yt*x[i]+y*xt[i];
23      y_t=y_t*x[i]+y*x_t[i];
24      y*=x[i];
25    }
26  }
27
28  void driver(vector<double>& x, double &y, vector<double>& g, vector<vector<double>>& H)
29    vector<double> x_t(x.size(),0);
30    vector<double> xt(x.size(),0);
31    vector<double> xt_t(x.size(),0);
32    double yt, y_t;
33    for (size_t i=0;i<x.size();i++) {
34      xt[i]=1;
35      for (size_t j=0;j<x.size();j++) {
36        x_t[j]=1;
37        f_tt(x,x_t,xt,xt_t,y,y_t,yt,H[i][j]);
38        x_t[j]=0;
39      }
40      g[i]=yt;
41      xt[i]=0;
42    }
43  }
44
45  int main(int c, char* v[]) {
46    assert(c==2); int n=atoi(v[1]); assert(n>0);
47    vector<double> x(n), g(n);
48    double y;
49    vector<vector<double>> H(n,vector<double>(n));
50    for (int i=0;i<n;i++) x[i]=cos(static_cast<double>(i));
51    driver(x,y,g,H);
52    cout << y << endl;
53    for (int i=0;i<n;i++) cout << g[i] << endl;
54    for (int i=0;i<n;i++) {
```

```
    for (int j=0;j<n;j++) cout << H[i][j] << " ";
    cout << endl;
  }
  return 0;
}
```

D.2.2 Adjoints

Listing 44: Second-Order Adjoint Product Reduction (Handwritten)

```
#include "std_includes.h"
using namespace std;

template<typename T>
void f_a_t(
    const vector<T>& x,
    const vector<T>& x_t,
    vector<T>& x_a,
    vector<T>& x_a_t,
    T& y,
    T& y_t,
    T& y_a,
    T& y_a_t
) {
  assert(x.size()>0);
  stack<T> tbr_t;
  stack<T> tbr;
  y_t=x_t[0];
  y=x[0];
  for (size_t i=1;i<x.size();i++) {
    tbr_t.push(y_t);
    tbr.push(y);
    y_t=y_t*x[i]+y*x_t[i];
    y*=x[i];
  }
  double ys_t=y_t;
  double ys=y;
  for (size_t i=x.size()-1;i>0;i--) {
    y_t=tbr_t.top(); tbr_t.pop();
    y=tbr.top(); tbr.pop();
    x_a_t[i]+=y_t*y_a+y*y_a_t;
    x_a[i]+=y*y_a;
    y_a_t=x_t[i]*y_a+x[i]*y_a_t;
    y_a=x[i]*y_a;
  }
  x_a_t[0]=y_a_t;
  x_a[0]=y_a;
```

```
38    y_a_t=0;
39    y_a=0;
40    y_t=ys_t;
41    y=ys;
42  }
43
44  void driver(vector<double>& x, double &y, vector<double>& g, vector<vector<double>>& h)
45    int n=x.size();
46    for (int i=0;i<n;i++) {
47      vector<double> x_t(n,0), x_a(n,0);
48      x_t[i]=1;
49      double y_a=1,y_a_t=0;
50      f_a_t(x,x_t,x_a,h[i],y,g[i],y_a,y_a_t);
51    }
52  }
53
54  int main(int c, char* v[]) {
55    assert(c==2); int n=atoi(v[1]); assert(n>0);
56    vector<double> x(n), g(n);
57    double y;
58    vector<vector<double>> H(n,vector<double>(n));
59    for (int i=0;i<n;i++) x[i]=cos(static_cast<double>(i));
60    driver(x,y,g,H);
61    cout << y << endl;
62    for (int i=0;i<n;i++) cout << g[i] << endl;
63    for (int i=0;i<n;i++) {
64      for (int j=0;j<n;j++) cout << H[i][j] << " ";
65      cout << endl;
66    }
67    return 0;
68  }
```

E Black Scholes PDE (Explicit Time Stepping)

E.1 First-Order AD

E.1.1 Tangents

Listing 45: Tangent Black Scholes PDE (Explicit Time Stepping; dco/c++)

```
1  # include "std_includes.h"
2  # include "f.h"
3
4  #include "dco.hpp"
5  typedef dco::gt1s<double>::type DCO_T;
6
7  typedef Matrix<DCO_T,Dynamic,1> DCO_VT;
```

```
8  typedef Matrix<double,Dynamic,1> VT;
9
10
11 VT driver(const VT& u, double e, double r, double sigma, int nt) {
12   int nx=u.size()+1;
13   VT g(nx+2);
14   DCO_VT u_(nx-1);
15   DCO_T e_=e, r_=r, sigma_=sigma;
16   // Delta
17   for (int i=0;i<nx-1;i++) {
18     for (int j=0;j<nx-1;j++) u_[j]=u[j];
19     dco::derivative(u_[i])=1;
20     f(u_,e_,r_,sigma_,nt);
21     g[i]=dco::derivative(u_[(nx-1)/2]);
22   }
23   // ???
24   for (int j=0;j<nx-1;j++) u_[j]=u[j];
25   dco::derivative(e_)=1;
26   f(u_,e_,r_,sigma_,nt);
27   g[nx-1]=dco::derivative(u_[(nx-1)/2]);
28   dco::derivative(e_)=0;
29   // Rho
30   for (int j=0;j<nx-1;j++) u_[j]=u[j];
31   dco::derivative(r_)=1;
32   f(u_,e_,r_,sigma_,nt);
33   g[nx]=dco::derivative(u_[(nx-1)/2]);
34   dco::derivative(r_)=0;
35   // Vega
36   for (int j=0;j<nx-1;j++) u_[j]=u[j];
37   dco::derivative(sigma_)=1;
38   f(u_,e_,r_,sigma_,nt);
39   g[nx+1]=dco::derivative(u_[(nx-1)/2]);
40   return g;
41 }
42
43 int main(int c, char* v[]) {
44   assert(c==3); int nx=atoi(v[1]), nt=atoi(v[2]);
45   const double e=0.5, r=0.03, sigma=0.5;
46   assert(nt>sigma*sigma*nx*nx);
47   assert(nt>(r*r)/(sigma*sigma));
48   VT u(nx-1); double u0=0;
49   for (int i=0;i<nx-1;i++) { u0=u0+1./nx; u[i]=max(u0-e,0.); }
50   VT greeks=driver(u,e,r,sigma,nt);
51   for (int i=0;i<nx-1;i++)
52     cout << "dVdu0[" << (i+1)*1./(nx-1) << "]=" << greeks[i] << endl;
53   cout << "dVde=" << greeks[nx-1] << endl;
```

```
54    cout << "dVdr=" << greeks[nx] << endl;
55    cout << "dVdsigma=" << greeks[nx+1] << endl;
56    return 0;
57  }
```

E.1.2 Adjoints

Listing 46: Adjoint Black Scholes PDE (Explicit Time Stepping; dco/c++)

```
1   # include "std_includes.h"
2   # include "f.h"
3
4   #include "dco.hpp"
5   typedef dco::ga1s<double> DCO_AM;
6   typedef DCO_AM::type DCO_A;
7   typedef DCO_AM::tape_t DCO_AM_TAPE;
8
9   typedef Matrix<DCO_A,Dynamic,1> DCO_VT;
10  typedef Matrix<double,Dynamic,1> VT;
11
12  VT driver(const VT& u, double e, double r, double sigma, int nt) {
13    int nx=u.size()+1;
14    VT g(nx+2);
15    DCO_VT u0_(nx-1);
16    for (int j=0;j<nx-1;j++) u0_[j]=u[j];
17    DCO_A e_=e, r_=r, sigma_=sigma;
18    DCO_AM::global_tape=DCO_AM_TAPE::create();
19    for (int j=0;j<nx-1;j++)
20      DCO_AM::global_tape->register_variable(u0_[j]);
21    DCO_AM::global_tape->register_variable(e_);
22    DCO_AM::global_tape->register_variable(r_);
23    DCO_AM::global_tape->register_variable(sigma_);
24    DCO_VT u_=u0_;
25    f(u_,e_,r_,sigma_,nt);
26    for (int j=0;j<nx-1;j++)
27      DCO_AM::global_tape->register_output_variable(u_[j]);
28    dco::derivative(u_[(nx-1)/2])=1;
29    DCO_AM::global_tape->interpret_adjoint();
30    // Delta
31    for (int j=0;j<nx-1;j++)
32      g[j]=dco::derivative(u0_[j]);
33    // ???
34    g[nx-1]=dco::derivative(e_);
35    // Rho
36    g[nx]=dco::derivative(r_);
37    // Vega
38    g[nx+1]=dco::derivative(sigma_);
```

```
39    return g;
40  }
41
42  int main(int c, char* v[]) {
43    assert(c==3); int nx=atoi(v[1]), nt=atoi(v[2]);
44    const double e=0.5, r=0.03, sigma=0.5;
45    assert(nt>sigma*sigma*nx*nx);
46    assert(nt>(r*r)/(sigma*sigma));
47    VT u(nx-1); double u0=0;
48    for (int i=0;i<nx-1;i++) { u0=u0+1./nx; u[i]=max(u0-e,0.); }
49    VT greeks=driver(u,e,r,sigma,nt);
50    for (int i=0;i<nx-1;i++)
51      cout << "dVdu0[" << (i+1)*1./(nx-1) << "]=" << greeks[i] << endl;
52    cout << "dVde=" << greeks[nx-1] << endl;
53    cout << "dVdr=" << greeks[nx] << endl;
54    cout << "dVdsigma=" << greeks[nx+1] << endl;
55    return 0;
56  }
```