Automatic Differentiation and Vibrato for the SDE in Finance

University of Paris VI, LPMA

Olivier Pironneau & Guillaume Sall & Gilles Pagès

LPMA-UPMC

March 13, 2017

Applications

- To compute sensitivities and trends (e.g. Greeks)
- To find the (local) minima/maxima of real valued functions
- And to characterize Pareto/Nash equilibria
- To implement Newton's root finding algorithm

The Tools

- By hand, Maple, Mathematica...
- By Vibrato or Malliavin Calculus
- By differentiating the program: ADOL-C, TAPENADE,
- By operator overloading in C++

Algorithmic, or automatic, differentiation (AD) is concerned with the accurate and efficient evaluation of derivatives for functions defined by computer programs. No truncation errors are incurred, and the resulting numerical derivative values can be used for all scientific computations that are based on linear, quadratic, or even higher order approximations to nonlinear scalar or vector functions.

A Griewank

Evaluating Derivatives - Principles and Techniques of Algorithmic Differentiation, SIAM 2000

Futures in Quantitative Finance

- A Financial asset S_t is modeled by

$$dX_t = X_t\Big(rdt + \sigma(t, X_t,)dW_t\Big), \quad X_0 = x \text{ known, } W_t \text{ Brownian motion}$$

- We write an "option" on S *i.e.* a contract with payment in the future, at "maturity" T. Typically, the European put option with payoff

$$(K - X_T)+ = \max(K - X_T), 0).$$

- In a a complete market (here local volatility market), the price of the contract is given by

$$P_t = e^{-r(T-t)}\mathbb{E}(K - X_T)_+.$$

- But, for such *vanilla* options, traders are not interested in the price: they are quoted on organized markets.
- They need for hedging purposes to know the sensitivities (greeks): 1^{st} and 2^{nd} derivatives of P_0 w.r. to K, T, S_0, r, σ.

- δ-hedge

$$\delta = \frac{\partial P_0}{\partial x}$$

- Γ (for the tracking error of δ-hedge of digital options)

$$\Gamma = \frac{\partial^2 P_0}{\partial x^2}$$

- The shock method: as for Γ the "simplest" approach is to

$$\Gamma = \frac{1}{h^2}\left(P_0|_{x+h} - 2P_0|_x + P_0|_{x-h}\right)$$

It costs 3 times the computation of the option but is it precise?

Tangent process *versus (?)* Malliavin Calculus

- Consider an option $P(x) = e^{-rT}\mathbb{E}[V(X_T^x)]$. with payoff V and spot price X_t^x satisfying an SDE with $X_0^x = x$. We want to compute

$$\delta := \partial_x \mathbb{E}[V(X_T^x)] \quad \text{and} \quad \Gamma := \partial_{xx}\mathbb{E}[V(X_T^x)].$$

- **Proposition.** One has

$$\partial_x \mathbb{E}[V(X_T^x)] = \underbrace{\mathbb{E}[V'(X_T^x)\partial_x X_T^x]}_{\text{pathwise differentiation}} = \underbrace{\mathbb{E}[V(X_T^x)\partial_x \log p]}_{\text{Int. by parts...}}.$$

- If the probability density p of X_T is known, the differentiation with respect to x could be done by an integration by parts:

$$\begin{aligned}\partial_x \mathbb{E}[V(X_T^x)] &= \partial_x \int_{\mathbf{R}^d} V(y)p_T(x,y)dy \\ &= \int_{\mathbf{R}^d} V(y)\partial_x \log(p_T)(x,y)p_T(x,y)dy = \mathbb{E}[V(X_T^x)\partial_x \log(p_T)(X_T^x)].\end{aligned}$$

- Otherwise, approximation by the Euler scheme, where $\bar{p}_T$ is explicit (in some sense) ...

- More generally, for two integrable random variables F and G, an integration by part is said to hold if there exists a random weight $H(F;G)$ such that for all smooth function Φ with compact support

$$\mathbb{E}[\Phi'(F)G] = \mathbb{E}[\Phi(F)H(F;G)].$$

Malliavin calculus provides a way to find the r.v. $H(X_T^x; \partial_x X_T^x)$.

- So, we have several methods:
  - Pathwise differentiation (when possible)
  - Shock method/finite difference (always possible)
  - Int. by Parts/ Malliavin
  - $\left[\text{Regularization + Pathwise (Vibrato Monte Carlo)}\right]$

Example: Malliavin Calculus Applied to the Greeks

For the 1^{st} and 2^{nd} derivatives with respect to initial conditions it gives:

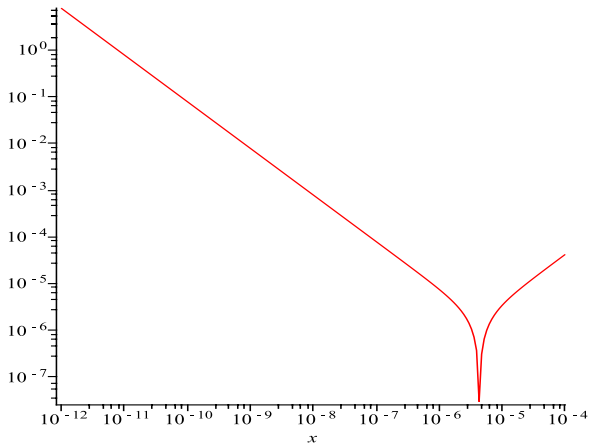
$$\delta = \mathbb{E}\left[e^{-rT}V(X_T)\frac{W_T}{x\sigma T}\right], \quad \Gamma = \mathbb{E}\left[\frac{e^{-rT}V(X_T)}{x^2 T\sigma}\left(\frac{W_T^2}{\sigma T} - W_T - \frac{1}{\sigma}\right)\right],$$

- Suppose a computer program implements in C

$$x = f(a), \quad \texttt{double f(double a).}$$

- Find x'_a? The solution is given by $x'_a = f'(a)$. How good is

```
double a=1., da=1e-4, dxda = (f(a+da)-f(a))/da;
```



- Example with $f(a) = sin(a), a = 1$. log-log plot of $|dxda - cos(1.)|$ (computed with Maple-14).

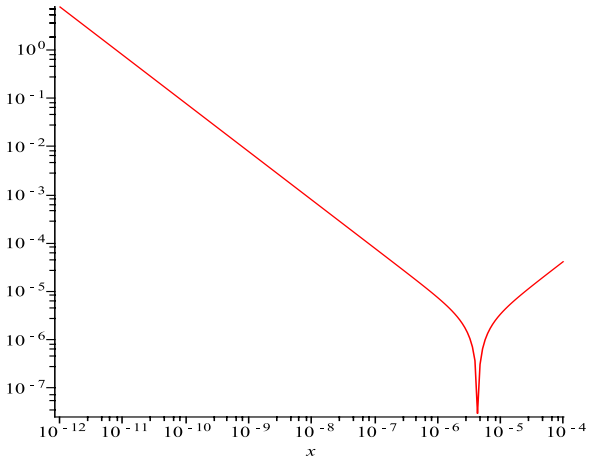
In Search of Derivatives (II)

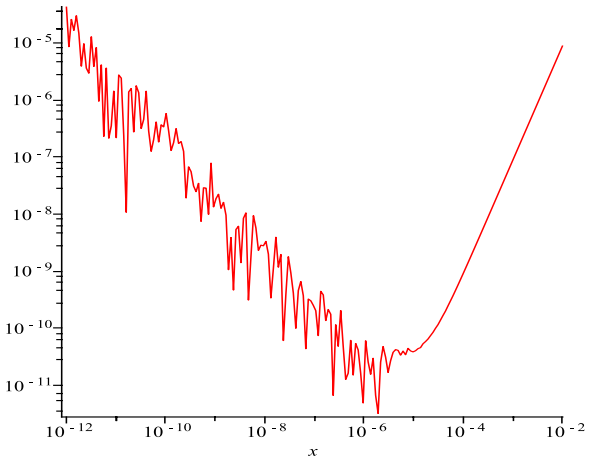
- Centered finite differences (no operator overloading in java: handy):

$$\frac{1}{2\delta a}(f(a+\delta a)-f(a-\delta a)) = f'(a) + f^{(3)}\frac{\delta a^2}{6} + o(\delta a^3)$$

- Set

```
double a=1., da=1e-4, dxda = (f(a+da)-f(a-da))/da/2
```





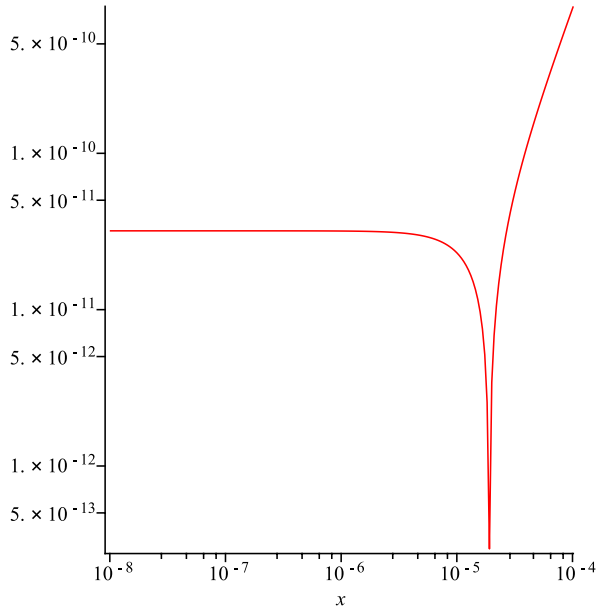
- Example : Asymmetric and centered finite difference with $f(a) = sin(a), a = 1.$ (computed with Maple-14).

In Search of Derivatives (III)

- Complex finite differences (if f is analytic)

$$Re\left[\frac{1}{\mathbf{i}\delta a}(f(a+\mathbf{i}\delta a)-f(a))\right] = f'(a) - f^{(3)}\frac{\delta a^2}{6} + o(\delta a^3)$$

```
double a=1., da=1e-4, dxda = ((f(a+I*da)-f(a))/da/I).R
```



- **Example:** $f(a) = sin(a), a = 1.$ (computed with Maple-14).
- It requires only one evaluation $\neq a$ and it is accurate for all $\delta a > 0$!
- Quiz: Why?

Principle of Automatic Differentiation (Forward (1965-70))

- **Fundamental Remark:** Connection between derivatives and differentials

  Let $J : (u, x, a) \mapsto J(u, x, a),\ u, x, a \in \mathbb{R}$ and its differential

$$\mathrm{d}J = J'_u(u, x, a)\mathrm{d}u + J'_x(u, x, a)\mathrm{d}x + J'_a(u, x, a)\mathrm{d}a$$

  where J'_u, J'_x, J'_a are the partial derivatives of J defined by

$$J'_u(u, x, a) = \lim_{\delta u \to 0} \frac{J(u + \delta u, x, a) - J(u, x, a)}{\delta u}, \text{ etc.}$$

  Setting $\mathrm{d}x = \mathrm{d}a = 0$ and $\mathrm{d}u = 1$, we get $J'_x(u, x, a) = \mathrm{d}J$.

- A simple example. Let $J(u) = |u - u_d|^2$, then its differential is

$$\mathrm{d}J = 2(u - u_d)(\mathrm{d}u - \mathrm{d}u_d), \qquad J'_u = 2(u - u_d)(1.0 - 0.0)$$

  Obviously J'_u is obtained by putting $\mathrm{d}u = 1,\ \ \mathrm{d}u_d = 0$.

A simple example (Step I)

- Let $J(u) = |u - u_d|^2$ and its differential

$$\mathrm{d}J = 2(u - u_d)(\mathrm{d}u - \mathrm{d}u_d).$$

- Now suppose that J is coded in C/C++ by

```
#include <iostream>
using namespace std;

double J(double u, double u_d){
    double z = u-u_d;
    double J = z*(u-u_d);
    return J;
}
int main(){  double u=2,u_d = 0.1;
    cout << J(u,u_d) << endl;
}
```

- A program which computes J and its differential can be obtained by writing above each differentiable line its differentiated form.

A simple example (Step II): the class ddouble

```
#include <iostream>
using namespace std;

class  ddouble { public: double val[2];
       ddouble(double a, double b){val[0]=a; val[1]=b;}
};
ddouble J(double u, double u_d, double du, double du_d){
    double dz = du - du_d;                    //AD
    double  z = u-u_d;
    double dJ = dz*(u-u_d) + z*(du - du_d); //AD
    double J= z*(u-u_d);
    return ddouble(J,dJ);
}
int main() {
    double u=2,u_d = 0.1;
    cout << J(u,u_d,1,0).val[1] << endl;
return 0;  }
```

- If `u-u_d` is implemented as `u.val[i]-u_d.val[i]`, i=0,1; then
  `double dz=du-du_d, z=u-u_d;` $\Leftrightarrow$ `ddouble u,u_d, z=u-u_d;`

Operator overloading

```
class  ddouble { public: double val[2];
    ddouble(double a=0, double b=0){ val[1]=b; val[0]=a; }
    ddouble operator=(const ddouble& a){
            val[1] = a.val[1]; val[0]=a.val[0];  return *this;
        }
    friend ddouble operator - (const ddouble& a, const ddouble& b){
            return ddouble(a.val[0] - b.val[0],a.val[1] - b.val[1]);
         }
    friend ddouble operator * (const ddouble& a, const ddouble& b){
            return ddouble(a.val[0] * b.val[0], a.val[1]*b.val[0] + a.val[0]*
            b.val[1]);
        }
};
ddouble J(ddouble u, ddouble u_d){
    ddouble z = u-u_d;
    ddouble J= z*(u-u_d);
    return J;
}
int main() {
    ddouble u=2., u_d = ddouble(0.1,1.);
    cout << J(u,u_d).val[1] << endl;
    return 0;  }
```

Conclusion: the initial program is untouched except that all `double` have been changed to `ddouble` and one variable has its `.val[1]=1`.

Non-differentiable Functions

- Automatic differentiation can be extended to handle derivatives of discontinuous functions by approximating the Dirac by

$$\delta^a(\xi) = \frac{1}{\sqrt{a\pi}} e^{\frac{\xi^2}{a}} \longrightarrow \delta_0 \quad \text{as} \quad a \to 0.$$

- Suppose we want to compute the K-greeks of

$$C(x, K) = \mathbb{E}\,(X_T - K)_+ = \int_{\mathbb{R}} (\xi - K)_+ p_{X_T}(\xi) d\xi.$$

- Then, if H denote the Heaviside function $H(u) = \mathbf{1}_{u \geq 0}$,

$$\partial_K C(x, K) = -\mathbb{E}\, H(X_T - K)$$

$$\begin{aligned} \text{and } \partial_{KK} C(x, K) &= p_{X_T}(K) = p_{X_T}(\xi) * \delta_0(K) \\ &= \lim_{a \to 0} \int p_{X_T}(\xi) * \delta^a(\xi - K) \mathrm{d}\xi \\ &\simeq \mathbb{E}\, \delta^a(X_T - K) \quad \text{as} \quad a \to 0. \end{aligned}$$

- This suggests to define the chain of automatic differentiation

$$\mathrm{ramp}(x) = x_+, \quad \mathrm{ramp}'(x) = H(x), \quad H'(x) \text{“}=\text{”} \delta^a$$

for a small enough $a > 0$.

- Consequently, for a risky asset X_T^x starting at $x$at $t = 0$

$$\delta(x) = \partial_x C(x, K) = \mathbb{E} H(X_T^x - K) \partial x X_T^x$$

and

$$\Gamma(x) = \partial_{xx} C(x, K) \simeq \mathbb{E}\left[\delta^a (X_T^x - K)(\partial_x X_T^x)^2 + H(X_T^x - K)\partial_{xx} X_T^x\right].$$

Forward AD: General Framework

- Let $x \in \mathbb{R}^n$ and $G(x) \in \mathbb{R}^m$ (instead of $J\dots$) and

$$G = g_p \circ g_{p-1} \cdots \circ g_2 \circ g_1, \qquad g_k : \mathbb{R}^m \to \mathbb{R}^m.$$

- Set $v_0 = x$ and $v_k = g_k(v_{k-1})$, $k = 1 : p-1$.

$$\mathrm{Jac}_G(x) = \mathrm{Jac}_{g_p}(v_{p-1})\mathrm{Jac}_{g_{p-1}}(v_{p-2}) \cdots \mathrm{Jac}_{g_1}(v_0).$$

- Define the `ddouble type` $(v_k, \dot{v}_k)$ where, $\dot{v}_k = \left[\frac{\partial v_k^i}{\partial v_0^j}\right]_{1 \le i \le m, 1 \le j \le n}$, $k = 1 = p$. Then

$$v_k = g_k(v_{k-1}) \quad \text{and} \quad \dot{v}_k = \mathrm{Jac}_{g_k}(v_{k-1})\dot{v}_{k-1}, \quad k = 1 : p.$$

- Let $\dot{v}_0 = e_j$ $j = 1 : n$. Then

$$v_p = G(x) \quad \text{and} \quad \dot{v}_p = \mathrm{Jac}_G(x)e_j = \frac{\partial G(x)}{\partial x_j} \in \mathbb{R}^m$$

- m "sweeps" are necessary to compute the j-th sensitivity,

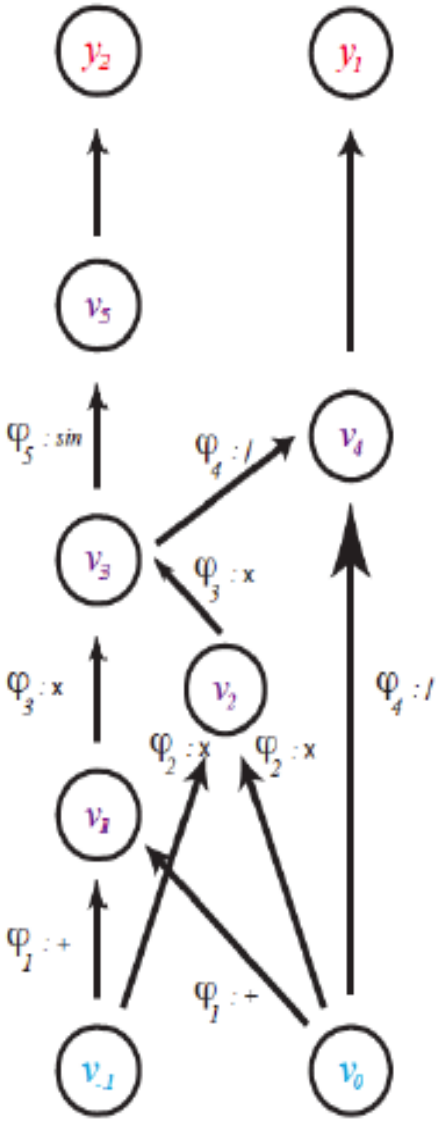
A toy example I

- **fonction** $F(x[\,]) : y[\,]$
  begin
  if $(x[1] \geqslant 2)$
  then $a = x[1] + x[2]$
  else $a = x[1] \cdot x[2]$
  fi
  for i in $1 \ldots 2$ **do**
  $a = a \cdot x[i]$
  od
  $y[1] = a/x[2]$
  $y[2] = sin(a)$
  return y
  end
- calcul en $x = (3; 1.5)$

v_{-1}	$=$	x_1	$=$	3.0
v_0	$=$	x_2	$=$	1.5
v_1	$=$	$v_{-1} + v_0$	$=$	4.5
v_2	$=$	$v_{-1} \cdot v_0$	$=$	4.5
v_3	$=$	$v_1 \cdot v_2$	$=$	20.25
v_4	$=$	v_3/v_0	$=$	13.5
v_5	$=$	$\sin(v_3)$	$=$	0.98552
y_1	$=$	v_4	$=$	13.5
y_2	$=$	v_5	$=$	0.98552

$$y = ((x_1 + x_2)x_1x_2/x_2; \sin(x_1 + x_2)x_1x_2))$$

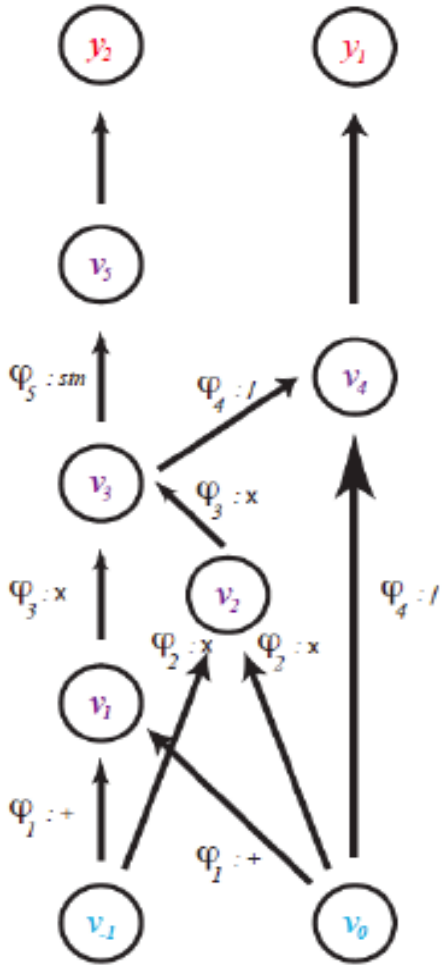
A toy example II



- chaque noeud = variable indépendante ou intermédiaire
- chaque arc = fonction mathématique élémentaire
- toute relation noeud $\leftrightarrow$ arc

$$v_i = \varphi_i(v_j)_{j \prec i} \quad i > 0, j < i$$

A toy example III



$$\dot{x} =^t \left(\frac{\partial x_1}{\partial x_1}; \frac{\partial x_1}{\partial x_2}\right) =^t (1;0)$$

v_{-1}	$=$	x_1	$=$	3.0
$\dot{v}_{-1}$	$=$	$\dot{x}_1$	$=$	1.0
v_0	$=$	x_2	$=$	1.5
$\dot{v}_0$	$=$	$\dot{x}_2$	$=$	0
v_1	$=$	$v_{-1} + v_0$	$=$	4.5
$\dot{v}_1$	$=$	$\dot{v}_{-1} + \dot{v}_0$	$=$	1
v_2	$=$	$v_{-1} \cdot v_0$	$=$	4.5
$\dot{v}_2$	$=$	$\dot{v}_{-1} \cdot v_0 + v_{-1} \cdot \dot{v}_0$	$=$	1.5
v_3	$=$	$v_1 \cdot v_2$	$=$	20.25
$\dot{v}_3$	$=$	$\dot{v}_1 \cdot v_2 + v_1 \cdot \dot{v}_2$	$=$	11.25
v_4	$=$	v_3/v_0	$=$	13.5
$\dot{v}_4$	$=$	$(\dot{v}_3 v_0 - v_3 \dot{v}_0)/v_0^2$	$=$	7.5
v_5	$=$	$\sin(v_3)$	$=$	0.98552
$\dot{v}_5$	$=$	$\cos(v_3)\dot{v}_3$	$=$	1.9072
y_1	$=$	v_4	$=$	13.5
$\dot{y}_1$	$=$	$\dot{v}_4$	$=$	7.5
y_2	$=$	v_5	$=$	0.98552
$\dot{y}_2$	$=$	$\dot{v}_5$	$=$	1.9072

- In forward AD the function and its partial derivatives are computed simultaneously.
- Empirical studies show that the evaluation time of partial/directional derivatives is around twice (2) or twice and a half (2.5) that of the function itself.
- It needs twice more memory allocation.
- Conclusion: If all sensitivities/directional derivatives are needed: $m \times n$ sweeps are necessary

Reverse Mode I

- One starts from the transposed identity

$$\mathrm{Jac}_G(x)^T = \mathrm{Jac}_{g_1}(v_0)^T \cdots \mathrm{Jac}_{g_p}(v_{p-1})^T$$

- Set $\bar{v}_k = \left[\frac{\partial v_p}{\partial v_k^i}\right]_{1\leq i\leq m}$, $k = 0 : p-1$. Then, $\bar{v}_p = Id_m$ and the chain differentiation rule yields

$$\bar{v}_k = \sum_{\ell=1}^{m} \frac{\partial v_p}{\partial v_{k+1}^\ell} \frac{\partial v_{k+1}^\ell}{\partial v_k^j} = \bar{v}_{k+1} \mathrm{Jac}_{g_{k+1}}(v_k)$$

i.e. the back propagation rule

$$\bar{v}_k^T = \mathrm{Jac}_{g_{k+1}}(v_k)^T \bar{v}_{k+1}^T.$$

- Finally, if for every $\bar{v}_p \in \mathbb{R}^m$,

$$\bar{v}_0^T = \mathrm{Jac}_{g_1}(v_0)^T \cdots \mathrm{Jac}_{g_p}(v_{p-1})^T \bar{v}_p^T = \mathrm{Jac}_G(x)^T \bar{v}_p^T.$$

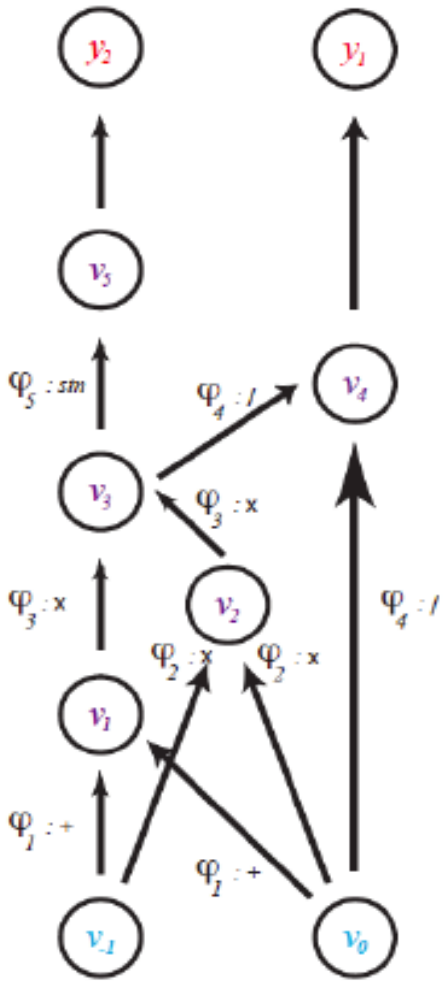
- Warning!! Propagation of values v_k remains forward:

$$v_k = g_k(v_{k-1}), \quad k = 1 : p,$$

while propagation of $\bar{v}_k$ is backward.

$$\bar{v}_k^T = \mathrm{Jac}_{g_{k+1}}(v_k)^T \bar{v}_{k+1}^T, \quad k = p-1 : 0.$$

A toy example III



$$\bar{y} =^t \left(\frac{\partial y_2}{\partial y_1}; \frac{\partial y_2}{\partial y_2}\right) =^t (0; 1)$$

$$
\begin{array}{rcccl}
v_{-1} & = & x_1 & = & 3.0 \\
v_0 & = & x_2 & = & 1.5 \\
v_1 & = & v_{-1} + v_0 & = & 4.5 \\
v_2 & = & v_{-1} \cdot v_0 & = & 4.5 \\
v_3 & = & v_1 \cdot v_2 & = & 20.25 \\
v_4 & = & v_3 / v_0 & = & 13.5 \\
v_5 & = & \sin(v_3) & = & 0.98552 \\
y_1 & = & v_4 & = & 13.5 \\
y_2 & = & v_5 & = & 0.98552 \\
\bar{v_5} & = & \bar{y_2} & = & 1.0 \\
\bar{v_4} & = & \bar{y_1} & = & 0 \\
\bar{v_3} & = & \bar{v_5} \cos(v_3) + \bar{v_4} / v_0 & = & 0.16952 \\
\bar{v_2} & = & \bar{v_3} v_1 & = & 0.76288 \\
\bar{v_1} & = & \bar{v_3} v_2 & = & 0.76288 \\
\bar{v_0} & = & \bar{v_{-1}} + \bar{v_2} v_{-1} - \bar{v_4} v_3 / v_0^2 & = & 3.05152 \\
\bar{v_{-1}} & = & \bar{v_1} + \bar{v_2} v_0 & = & 1.9072 \\
\bar{x_2} & = & \bar{v_0} & = & 3.05152 \\
\bar{x_1} & = & \bar{v_{-1}} & = & 1.9072
\end{array}
$$

First conclusions

- Reverse/adjoint mode is best suited for $G : \mathbb{R}^n \to \mathbb{R}^m$, $n \gg m$.
- Forward mode is best suited for $G : \mathbb{R}^n \to \mathbb{R}^m$, $n \ll m$.
- For Greeks, $m = 1 \Longrightarrow$ the reverse mode is recommended... but limited by memory constraints (even if $n \approx 5$).
- Forward and reverse mode represent just two possible (extreme) ways of recursing through the chain rule.
- There exists an optimal recursion, but finding it is most likely an NP-hard problem.

How to implement it?

- (AD) provides path wise differentiation formula

$$\partial_x \mathbb{E}[V(X_T^x)] = \underbrace{\mathbb{E}[V'(X_T^x)\partial_x X_T^x]}_{\text{pathwise differentiation}}$$

- . . . that can be computed by a regular Monte Carlo simulation.

Conclusion (on *AD*)

- AD is a simple and powerful tool
- Good to know what happens behind the scene
- Fancy C++ for professional implementation
- OK in Python, C#, a little in Fortran 95, not in Java
- Reverse mode is only for the professional

Bibliography (AD)

- "Evaluating Derivatives : Principles and Techniques of Algorithmic Differentiation" - A. Griewank - SIAM 2008
- "La différentiation automatique de fonctions représentées par des programmes" - JC. Gilbert, G. Le Vey, J. Masse - RR 1557 Novembre 1991 - INRIA Rocquencourt
- Différentiation Automatique de programmes" - L. Hascoet (INRIA Sophia Antipolis, projet TROPICS) - Séminaire Ingéniérie mathématique Cnam - septembre 2010
- "Computing Adjoints by Automatic Differentiation with TAPENADE" - L Hascoet, R. -M. Greborio, V Pascual - Conference proceeding, Ecole INRIA-CEA-EDF "Problèmes non-linéaires appliqués", Springer, 2005
- www.autodiff.org

How to use “Malliavin” weights ?

Vibrato (McGiles)

- Let θ be a parameter and the aim is to compute $\partial_\theta \mathbb{E}[V(X_T)]$

$$dX_t = b(\theta, X_t)\,dt + \sigma(\theta, X_t)dW_t,\ X_0 = x. \tag{1}$$

- Consider Euler explicit scheme X_t (Markov chain):

$$\bar{X}_k^n = \bar{X}_{k-1}^n + b(\theta, \bar{X}_{k-1}^n)h + \sigma(\theta, \bar{X}_{k-1}^n)\sqrt{h}\,Z_k, \quad \bar{X}_0 = x, \quad k = 1, \ldots, n,$$

where $W_{t_k^n} - W_{t_{k-1}^n} = \sqrt{h}\,Z_k$.

- First idea: Note that

$$\bar{X}_n^n = \mu_{n-1}(\theta) + \sigma_{n-1}(\theta)\sqrt{h}Z_n$$

with

$$\mu_{n-1}(\theta) = \bar{X}_{n-1}^n(\theta) + b(\theta, \bar{X}_{n-1}^n(\theta))h \ \text{ and } \ \sigma_{n-1}(\theta) = \sigma(\theta, \bar{X}_{n-1}^n(\theta)).$$

- Hence

$$\mathbb{E}\left[V(\bar{X}_n^n)\right] = \mathbb{E}\left[\mathbb{E}\left[V(\bar{X}_n^n) \mid \mathcal{F}_{n-1}\right]\right] = \mathbb{E}\left[\mathbb{E}\left[V(\bar{X}_n^n) \mid \bar{X}_{n-1}^n\right]\right].$$

Vibrato (McGiles)

- Then

$$\frac{\partial}{\partial \theta_i}\mathbb{E}[V(\bar{X}_n^n(\theta))] = \mathbb{E}\left[\frac{\partial}{\partial \theta_i}\left\{\mathbb{E}[V(\mu+\sigma Z\sqrt{h})]_{\left|\begin{smallmatrix}\mu = \mu_{\boldsymbol{n-1}}(\theta)\\ \sigma = \sigma_{\boldsymbol{n-1}}(\theta)\end{smallmatrix}\right.}\right\}\right]. \tag{2}$$

- $\mathbb{E}\left[V(\bar{X}_n^n(\theta))\right]$ is a smooth function (regularization effect of Z) but unknown!
- The last time step has constant coefficient, so we have an explicit solution so Malliavin weights formal still applies..

Vibrato(II)

- **Proposition**

$$\frac{\partial}{\partial\theta}\mathbb{E}[V(\bar{X}_n^n(\theta))] = \mathbb{E}\left[\frac{\partial}{\partial\theta}\left\{\mathbb{E}_z[V(\mu+\sigma Z\sqrt{h})]\right\}_{\substack{\mu=\mu_{n-1}(\theta)\\ \sigma=\sigma_{n-1}(\theta)}}\right]$$

$$= \mathbb{E}\left[\frac{1}{\sqrt{h}}\frac{\partial\mu}{\partial\theta}\cdot\mathbb{E}_z\left[V(\mu+\sigma Z\sqrt{h})\sigma^{-T}Z\right]\Big|_{\substack{\mu=\mu_{n-1}(\theta)\\ \sigma=\sigma_{n-1}(\theta)}}\right.$$

$$+\frac{1}{2}\frac{\partial(\sigma\sigma^T)}{\partial\theta}:\mathbb{E}_z\left[V(\mu+\sigma Z\sqrt{h})\sigma^{-T}(ZZ^T-I)\sigma^{-1}\right]\Big|_{\mu}$$

where $\mu_{n-1}(\theta) = \bar{X}_{n-1}^n(\theta) + b(\theta, \bar{X}_{n-1}^n(\theta))h$ and $\sigma_{n-1}(\theta) = \sigma(\theta, \bar{X}_{n-1}^n(\theta))$.

- Works great but for 2^{nd} derivatives Vibrato is twice as messy and diffusive.

Monte Carlo Vibrato

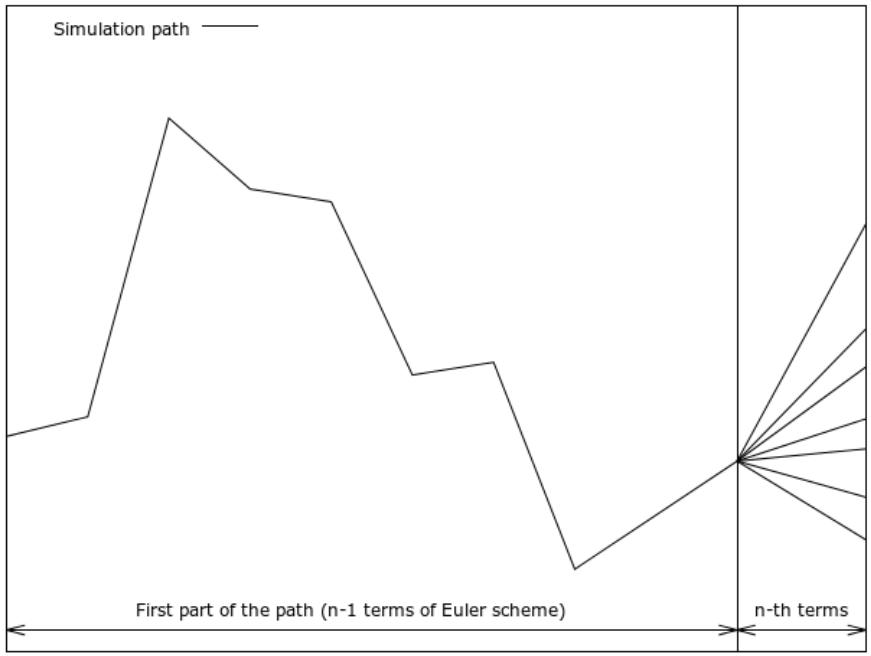


Figure: Gamma. *Schematics of Vibrato..*

- It is a nested Monte Carlo: M_X "outer paths" to compute the outer expectations based on $\mu_{n-1}(\theta)$, $\sigma_{n-1}(\theta)$.
- M_Z paths for the inner expectation.
- $M_Z \ll M_X$ (often $M_Z = 1$ or 2 !)

Vibrato+AD on a plain vanilla option

- For *AD*, we use the library adept by R.J. Howen which is both direct and reverse mode for C++

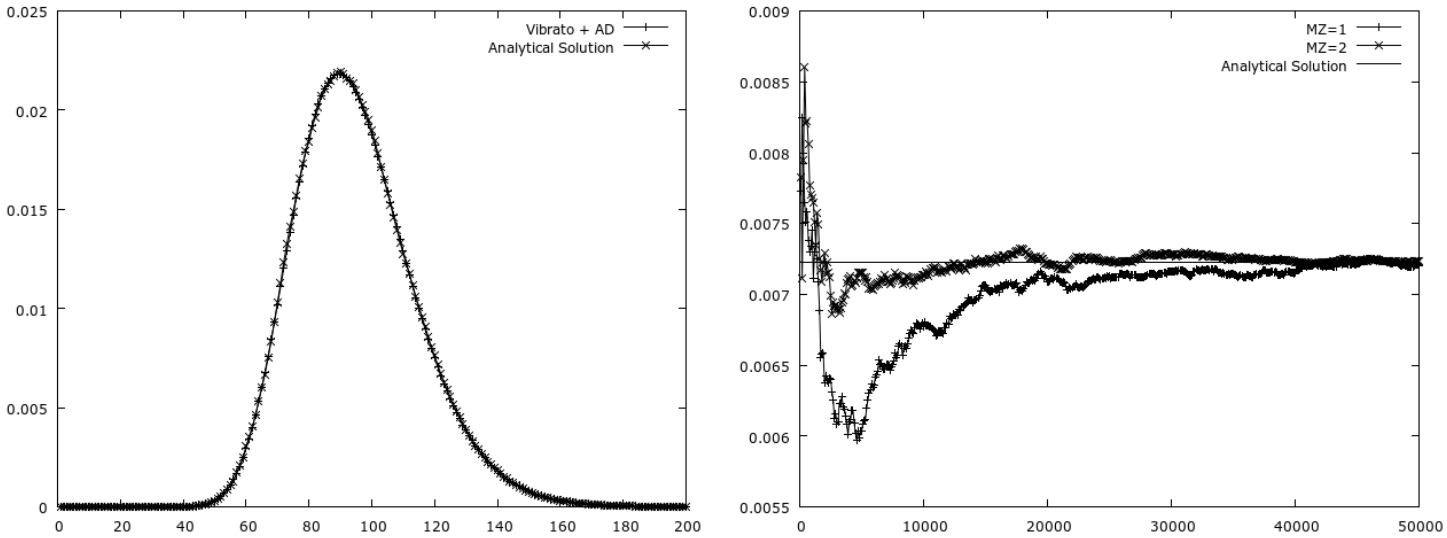


Figure: *On the left the Gamma versus Price is displayed when computed by Vibrato + AD; the analytical exact Gamma is also displayed; both curves overlap. On the right, the convergence history at one point* $X_0 = 120$ *is displayed with respect to the number of outer Monte Carlo samples* M_X*. This is done for two values of* M_Z *(the number of the final time step),* $M_Z = 1$ *(low curve) and* $M_Z = 2$ *(upper curve).*

Vibrato+AD: Result on a Basket

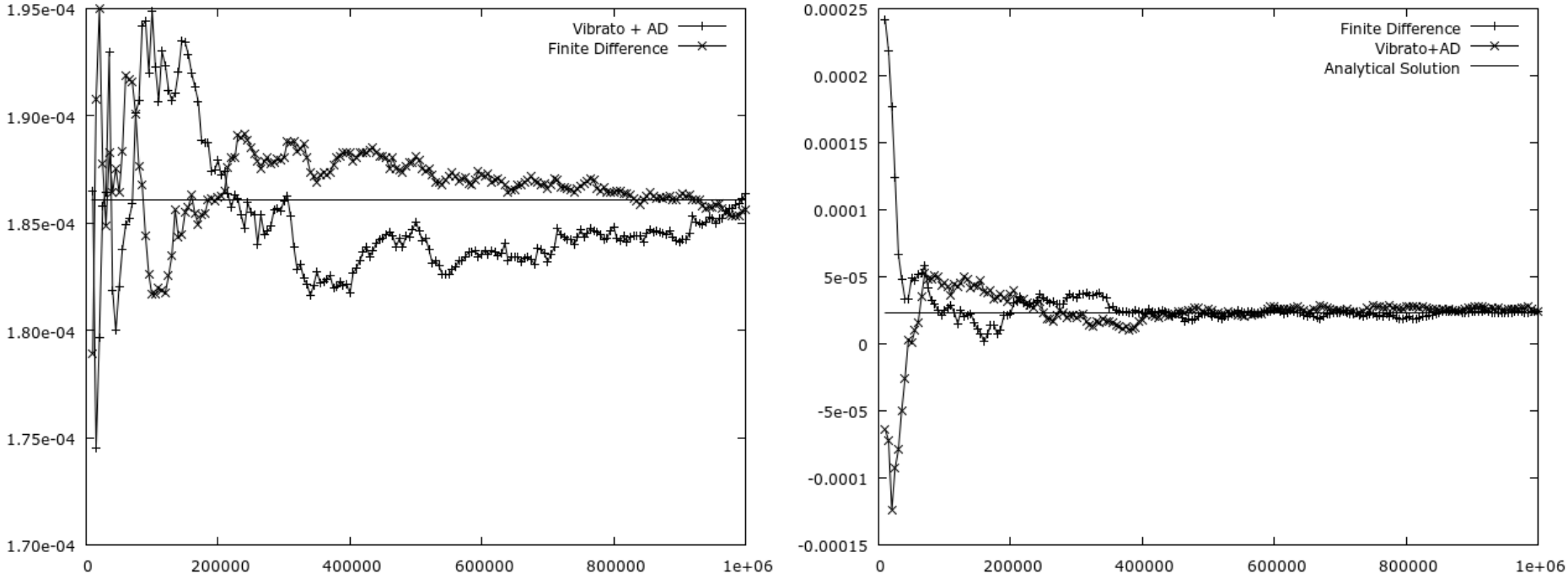


Figure: Convergence of the computation of the Gamma of a Basket option in $d = 4$ (left) and $d = 7$ (right) via Vibrato plus Automatic Differentiation on Monte Carlo and via Finite differences, versus the number of simulation paths.

Vibrato of Vibrato (on a plain vanilla Call)

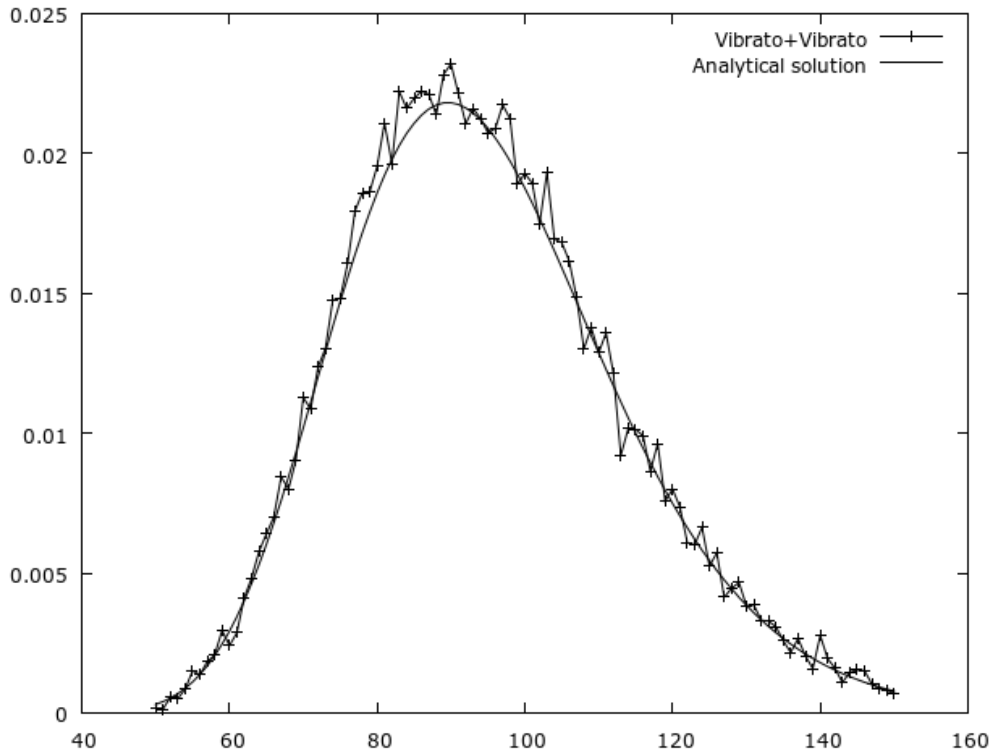


Figure: Gamma. *The gamma versus price is displayed when computed by Vibrato plus Vibrato; the analytical exact Gamma is also displayed.*

- $X_0 \in (0, 200]$, $K = 100$, $\sigma = 20\%$ and $r = 5\%$, $T = 1$ year.
- Monte Carlo parameters: $M = 10,000$ trials and $n = 25$ time steps.

Comparison of Malliavin Calculus and Vibrato+AD

- Malliavin calculus is also based on an integration by parts.
- For a European call when σ and b are constant (B-S model!), the Malliavin estimator for the Gamma is:

$$\tilde{\Gamma} = e^{-rT}\mathbb{E}\left[(X_T^x - K)^+ \frac{1}{x^2\sigma T}\left(\frac{W_T^2}{\sigma T} - \frac{1}{\sigma} - W_T\right)\right] \tag{3}$$

- The main problem is that it is singular at $T = 0$ and the formulae are different and not always easy to obtain for different futures.

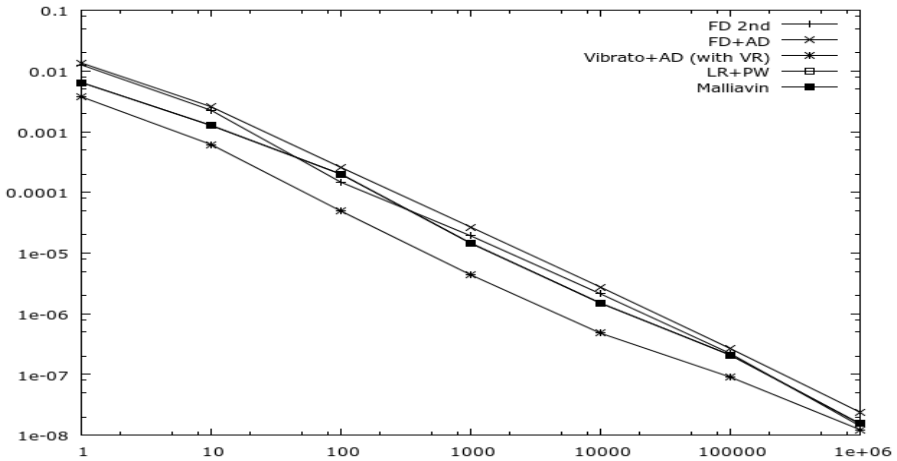


Figure: *L_2-error with LR-PW, 2^{nd} FD, FD+AD, Malliavin, Vibrato+AD*

Thank you for the invitation