

Automatic Differentiation & Second Sensitivities in Quantitative Finance

Olivier Pironneau¹
+ Gilles Pagès & Guillaume Sall

Sorbonne Universities, UPMC (Paris VI), Laboratoire J.-L. Lions (LJLL)

Monte-Carlo conf. July 16



Greeks for Futures

A Financial asset X_t modeled by

$$dX_t = X_t \left(r(t)dt + \sigma(X_t, t)dW_t \right), \quad X_0 \text{ given, } W_t \text{ Brownian}$$

A European put option $V_t = e^{-r(T-t)}\mathbb{E}(K - X_T)^+$

Compute the 1st and 2nd derivatives of V_0 w.r. to K, T, X_0, r, σ , such as

$$\Gamma = \frac{\partial^2 V_0}{\partial X_0^2} \text{ Need also to compute } \frac{\partial^2}{\partial X_0^2} \int_0^\infty f(x)(K - x)^+ dx$$

The most straightforward approach is to

$$\Gamma = \frac{1}{h^2} \left(V_0|_{x_0+h} - 2V_0|_{x_0} + V_0|_{x_0-h} \right)$$

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



In Search of Automatic Differentiation

Let $x = f(a)$ be programmed in C by `double f(double a)`. Find x'_a ? by

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

;

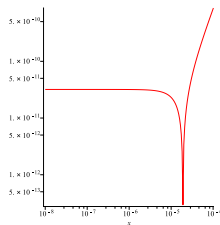
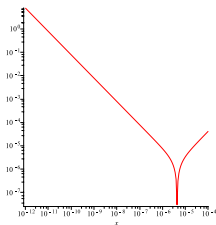


Figure: Example: $f(a) = \sin(a)$, $a = 1$. log-log plot of $|dxda - \cos(1.)|$
Complex finite differences

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

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



Second Derivatives

Example: $f(x) = (1 - x)^+ \Rightarrow f'(x) = -\mathbf{1}_{x < 1}, \quad f''(x) = \delta(1 - x)$

$f(x) = (1 - x)^+ \Rightarrow$ Maple code: `plot(max(1-x,0))`

`plot((max(1-x-1e-8,0)-max(1-x,0))*1e8);`

`plot((max(1-x-3.5e-6,0)-2*max(1-x,0)+max(1-x+3.5e-6,0))*1e12)`

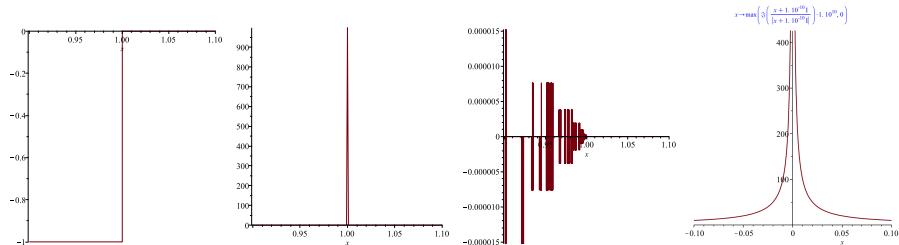


Figure: 1st derivative with $\delta x = 10^{-8}$, 2nd with $\delta x = 10^{-3}$ and $\delta x = 3.510^{-6}$.
Far right with complex difference



Malliavin Calculus

Consider the payoff V of a future with spot price $X_t(x)$ satisfying an SDE with $X_0 = x$.

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

Proposition.

If the probability density p of X_T is known,

$$\partial_x \mathbb{E}[V(X_T)] = \partial_x \int_{R^d} V(s) p(s, x) ds = \int_{R^d} V(s) \partial_x \log(p) p ds = \mathbb{E}[V(X_T) \partial_x \log(p)].$$

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

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

Malliavin calculus gives a way to find **Weight Function** $H(X_T; \partial_x X_T)$.



Malliavin Calculus Applied to the δ and Γ of a Vanilla Put

For the 1st and 2nd derivatives with respect to initial conditions it gives

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

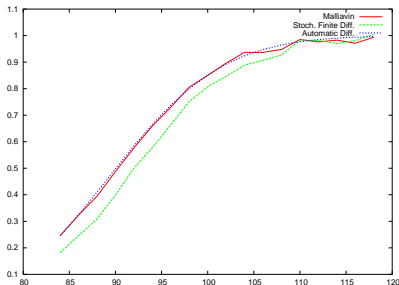


Figure: The δ with $\sigma = 0.1$, $r = 0.1$, $T = 1$, $K = 100$ for $X_0 \in (84, 118)$. Comparison with stochastic centered finite differences, and AD on the PDE. Nb of M-C path is 50000 with 100 time step.

It works so long as the weight functions are known and T not too small



Vibrato (McGiles)

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

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

Consider an Euler explicit scheme X_t :

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

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

First idea: write $\mathbb{E}[V(\bar{X}_n^n)] = \mathbb{E}[\mathbb{E}[V(\bar{X}_n^n) | \bar{X}_{n-1}^n]]$.

Note that $\bar{X}_n^n = \mu_{n-1}(\theta) + \sigma_{n-1}(\theta)Z_n\sqrt{h}$ with
 $\mu_{n-1}(\theta) = \bar{X}_{n-1}^n(\theta) + b(\theta, \bar{X}_{n-1}^n(\theta))h$
 $\sigma_{n-1}(\theta) = \sigma(\theta, \bar{X}_{n-1}^n(\theta))$. Then

$$\frac{\partial}{\partial \theta_i} \mathbb{E}[V(\bar{X}_n^n(\theta))] = \mathbb{E}_z \left[\frac{\partial}{\partial \theta_i} \left\{ \mathbb{E}[V(\mu + \sigma Z\sqrt{h})] \right\}_{\substack{\mu = \mu_{n-1}(\theta) \\ \sigma = \sigma_{n-1}(\theta)}} \right] \quad (2)$$

The last time step sees constant coefficient, so we have an explicit solution



Vibrato(II)

The last time step has constant coefficient, the density p is known and
Second idea:

$$\frac{\partial}{\partial \theta} \left[\mathbb{E}[V(X(\theta))] \right] = \int_{\mathbb{R}^d} V(y) \frac{\partial \log p}{\partial \theta}(\theta, y) p(\theta, y) dy = \mathbb{E} \left[V(X(\theta)) \frac{\partial \log p}{\partial \theta}(\theta, X(\theta)) \right]$$

Proposition

$$\begin{aligned} \frac{\partial}{\partial \theta} \mathbb{E}[V(\bar{X}_n^n(\theta))] &= \mathbb{E} \left[\frac{1}{\sqrt{h}} \frac{\partial \mu}{\partial \theta} \cdot \mathbb{E}_Z \left[V(\mu + \sigma Z \sqrt{h}) \sigma^{-T} Z \right] \right. \\ &\quad \left. + \frac{1}{2} \frac{\partial(\sigma \sigma^T)}{\partial \theta} : \mathbb{E}_Z \left[V(\mu + \sigma Z \sqrt{h}) \sigma^{-T} (ZZ^T - I) \sigma^{-1} \right] \right]_{\substack{\mu = \mu_{n-1}(\theta) \\ \sigma = \sigma_{n-1}(\theta)}} \end{aligned}$$

where $\mu_{n-1}(\theta) = \bar{X}_{n-1}^n(\theta) + b(\theta, \bar{X}_{n-1}^n(\theta))h$ and $\sigma_{n-1}(\theta) = \sigma(\theta, \bar{X}_{n-1}^n(\theta))$.
In the non constant case the tangent process $Y_t = \partial_\theta X_t$ is involved.

Note that V is not differentiated! CPU in the non constant case is twice the evaluation of $V(X_0)$.



Vibrato of Vibrato = Derivative of Vibrato formula

Works great but for 2nd derivatives Vibrato is twice as involve (and diffusive).

Proposition

$$\begin{aligned} \frac{\partial^2}{\partial \theta^2} \mathbb{E}[V(X_T)] = & \mathbb{E} \left[\frac{\partial^2 \mu}{\partial \theta^2} \mathbb{E} \left[V(\mu + \sigma \sqrt{h} Z) \frac{Z}{\sigma \sqrt{h}} \right] + \left(\frac{\partial \mu}{\partial \theta} \right)^2 \mathbb{E} \left[V(\mu + \sigma \sqrt{h} Z) \frac{Z^2 - 1}{\sigma^2 h} \right] + \left(\frac{\partial \sigma}{\partial \theta} \right)^2 \mathbb{E} \left[V(\mu + \sigma \sqrt{h} Z) \right. \right. \\ & \left. \left. \frac{Z^4 - 5Z^2 + 2}{\sigma^2 h} \right] + \frac{\partial^2 \sigma}{\partial \theta^2} \mathbb{E} \left[V(\mu + \sigma \sqrt{h} Z) \frac{Z^2 - 1}{\sigma \sqrt{h}} \right] + 2 \frac{\partial \mu}{\partial \theta} \frac{\partial \sigma}{\partial \theta} \mathbb{E} \left[V(\mu + \sigma \sqrt{h} Z) \frac{Z^3 - 3Z}{\sigma^2 h} \right] \right] \quad (3) \end{aligned}$$

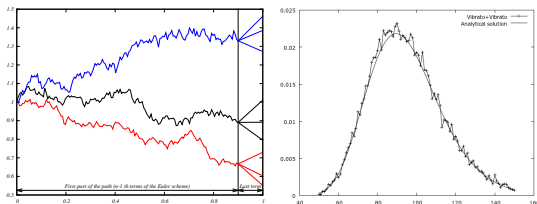


Figure: On the left the Vibrato idea. On the right the Γ versus price computed by Vibrato + Vibrato compared with exact Γ . $K = 100$, $\sigma = 20\%$ and $r = 5\%$, $T = 1$ year. $X_0 \in (1, 200)$, 10,000 M-C paths and 25 time steps.

Antithetic Variance reduction can be worked out (see paper)



Principle of AD in Forward Mode

Fundamental: Relation between derivatives and differentials

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

$$dJ = J'_u(u, x, a)du + J'_x(u, x, a)dx + J'_a(u, x, a)da$$

By taking $dx = da = 0$ and $du = 1$, we have $J'_u(u, x, a) = dJ$.

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

$$dJ = 2(u - u_d)(du - du_d), \quad J'_u = 2(u - u_d)(1.0 - 0.0)$$

obtained by putting $du = 1$, $du_d = 0$ in the first line of code.

So always manipulate the variable and its differential together ($x \triangleright dx$)
(Barak Pearlmutter[1]).



A simple example

Fundamental: Each line of code is differentiable by hand exactly.

Let $J(u) = |u - u_d|^2$, then its differential is $dJ = 2(u - u_d)(du - du_d)$.

Now suppose that J is programmed 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 (cont)

```
#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; }
```

Note that if $u-u_d$ is implemented as $u.val[i]-u_d.val[i], i=1;2$ then

$double dz=du-du_d, z=u-u_d; \Leftrightarrow ddouble u,u_d, z=u-u_d;$



The class ddouble

```
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.



Limitations

- Extend to M^{th} -derivative and N parameters but $\text{CPU} \sim O(N \times M)$.
- Operator overloading does not exist in all languages (not in java or in C)
- \sqrt{x} is not differentiable at $x = 0$
- Root finding by Newton algorithm is iterative:

$$f(x, \alpha) = 0 \Rightarrow x' f'_x + f'_\alpha = 0 \Rightarrow x' = -\frac{f'_\alpha}{f'_x}$$

```
program newtonest
x=0.0;
alpha=0.5
call newton(x,10,alpha)
write(*,*) x
end

subroutine newton(x,n,alpha)
do i=1,n
  f = x-alpha*cos(x)
  fp= 1+alpha*sin(x)
  x=x-f/fp
enddo
return
end
```



Non Differentiable Functions

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

$$(\sqrt{x})' \approx \frac{1}{2\sqrt{x+\epsilon}}, \quad (\mathbf{1}_{x>0})' \approx \delta^a(x) = \frac{1}{\sqrt{a\pi}} e^{-\frac{x^2}{a}}$$

If f is discontinuous at $x = z$ and smooth elsewhere. Let $H(x) = \mathbf{1}_{x>0}$,

$$f(x) = f^+(x)H(x-z) + f^-(x)(1-H(x-z))$$

$$f'_z(x) = (f^+)'_z(x)H(x-z) + (f^-)'_z(x)(1-H(x-z)) - (f^+(z) - f^-(z))\delta(x-z)$$

Add to the library δ^a and add $H(x)$, $\text{ramp}(x)=x^+$ with derivatives $\delta^a(x)$ and $H(x)$ and in the computer program don't write $\max(K-x,0)$ but write $\text{ramp}(K-x)$; then the second derivative in K will be $\delta^a(K-x)$ and

$$\int_0^\infty f(x)(K-x)^+ dx \approx \frac{1}{N} \sum f(\xi_i)\delta^a(K-\xi_i)$$

where ξ_i are the N quadrature or Monte-Carlo points used by the programmer to approximate the integral.



History and Pointers

- The creator of Fortran in the sixties were aware of the capacity of compilers to differentiate a program
- The creator of modern AD is Andreas Griewank (Now in Germany)
- Adol-C is based on operator overloading; it is open source
- INRIA started with odyssee and rewrote it to tapenade (web based)
- Similar functions available in NAG and fadBAD++
- **ADePT** by Robin Hogan, University of Reading. Easy to use, efficient, but not maintained.

Uwe Naumann: *The art of differentiating computer programs*. SIAM series. Pittsburg (2012).

Andreas Griewank and Alan Walther: *Evaluating Derivatives: Principle and Techniques of Algorithmic Differentiation*. SIAM series. (2008).



Comparison Vibrato+AD (VAD) with Malliavin Calculus

Malliavin estimator for the Gamma is:

$$\tilde{\Gamma} = e^{-rT} \mathbb{E} \left[(X_T - K)^+ \frac{1}{X_0^2 \sigma T} \left(\frac{W_T^2}{\sigma T} - \frac{1}{\sigma} - W_T \right) \right] \quad (4)$$

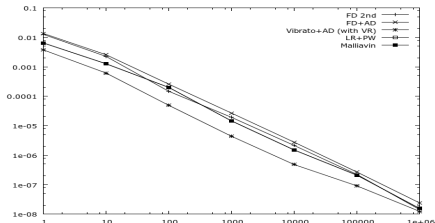


Figure: L_2 -error with LR-PW, 2nd FD, 1st FD+AD, Malliavin, Vibrato+AD

CPU of **VAD** is similar to FD+AD, 2nd FD and roughly twice that of Malliavin. The difference is in the precision (variance)



Vibrato+AD on a plain vanilla option

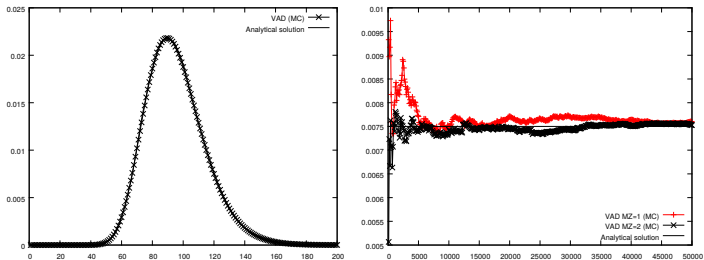


Figure: On the left the Gamma versus Price is displayed when computed by Vibrato + AD; the analytical exact Gamma is also displayed; both curves overlap. On the right, the convergence history at one point $X_0 = 120$ is displayed with respect to the number of Monte Carlo samples M_W . This is done for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

VAD on a Basket

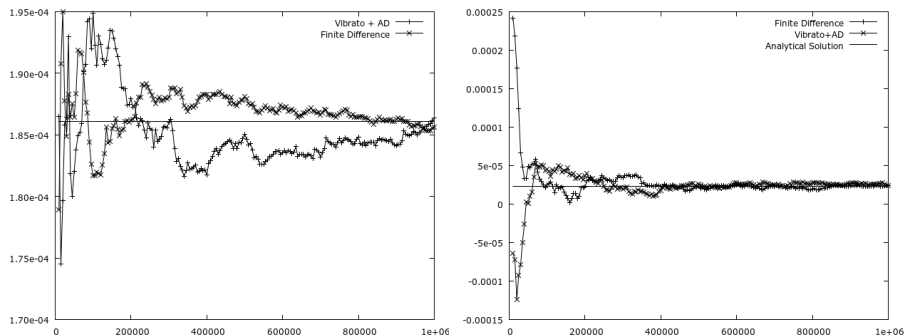


Figure: Convergence of the computation of the Gamma of a Basket option in $d = 4$ (left) and $d = 7$ (right) by VAD and by Finite differences, versus the number of simulation paths.

Other Derivatives

The Vanna is $\partial_{X_0, \sigma} V$

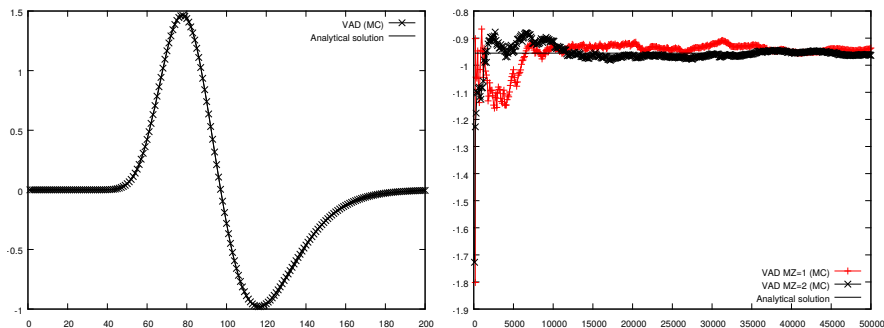


Figure: On the left the Vanna versus Price is displayed when computed by VAD; the analytical exact Vanna is also displayed; both curves overlap. On the right, the convergence history at one point $X_0 = 120$ is displayed with respect to the number of Monte Carlo samples M_W . This is done for two values of M_Z , $M_Z = 1$ (red curve) and $M_Z = 2$ (black curve).



Third Derivatives by Vibrato of Vibrato + AD (VVAD)

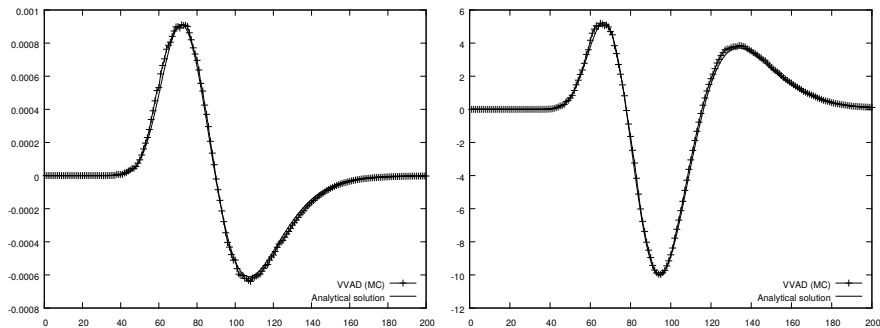



Figure: On the left $\partial^3 V / \partial X_0^3$ versus Price is displayed when computed by VVAD; the analytical exact curve is also displayed. On the right, the same for the sensitivity of the Vanna with respect to changes in interest rate ($\partial^3 V / \partial X_0 \partial \sigma \partial r$).

Vibrato+Reverse Mode AD (VRAD)

$$\begin{pmatrix} \frac{\partial^2 V}{\partial X_0^2} & \frac{\partial^2 V}{\partial v \partial X_0} & \frac{\partial^2 V}{\partial r \partial X_0} & \frac{\partial^2 V}{\partial T \partial X_0} \\ \frac{\partial X_0 \partial \sigma}{\partial^2 V} & \frac{\partial \sigma^2}{\partial^2 V} & \frac{\partial v \partial r}{\partial^2 V} & \frac{\partial T \partial v}{\partial^2 V} \\ \frac{\partial X_0 \partial r}{\partial^2 V} & \frac{\partial v \partial r}{\partial^2 V} & \frac{\partial r^2}{\partial^2 V} & \frac{\partial T \partial r}{\partial^2 V} \\ \frac{\partial X_0 \partial T}{\partial^2 V} & \frac{\partial v \partial T}{\partial^2 V} & \frac{\partial r \partial T}{\partial^2 V} & \frac{\partial T^2}{\partial^2 V} \end{pmatrix}. \quad (5)$$

Mode	FD (MC)	VRAD (MC)
Time (sec)	2.01	0.47

Table: CPU time to compute the Hessian matrix of a standard European Call option (considering X_0 , σ , r , T as variables) in the Black-Scholes model.

Hi-tech implementation of `class ddouble` using traits shows that there is no need to go to reverse mode AD when the number of partial derivatives to be computed is less than 20. (Thesis of N. DiCesaré (now at Natixis)) 

Greeks for American Options by AD

Vibrato + AD can be used within the Longstaff-Schwarz algorithm.

S	σ	T	Price	Stdr Error	δ -Vibrato	Stdr Error	Γ -Vibrato	Stdr Error
36	0.2	1	4.46289	0.013	0.68123	1.820e-3	0.06745	6.947e-5
36	0.2	2	4.81523	0.016	0.59934	1.813e-3	0.06398	6.846e-5
36	0.4	1	7.07985	0.016	0.51187	1.674e-3	0.03546	4.852e-5
36	0.4	2	8.45612	0.024	0.44102	1.488e-3	0.02591	5.023e-5
38	0.2	1	3.23324	0.013	0.53063	1.821e-3	0.07219	1.198e-4
38	0.2	2	3.72705	0.015	0.46732	1.669e-3	0.05789	1.111e-4
38	0.4	1	6.11209	0.016	0.45079	1.453e-3	0.03081	5.465e-5
38	0.4	2	7.61031	0.025	0.39503	1.922e-3	0.02342	4.827e-5
40	0.2	1	2.30565	0.012	0.40780	1.880e-3	0.05954	1.213e-4
40	0.2	2	2.86072	0.014	0.39266	1.747e-3	0.04567	5.175e-4

Linking to ddouble, can give derivatives with respect to any parameters.

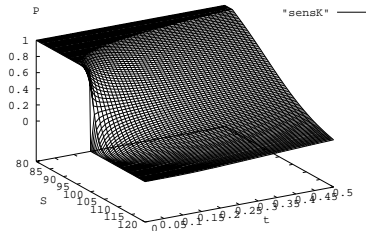


Figure: Sensitivity w.r. to K of an American Put vs asset price and $T-t$



Reverse Mode

Consider $J(u, x)$ where $A(u)x = f(u)$. It models a typical computer program where x are the intermediary variables. In computer programs A may be nonlinear but it is lower triangular. Then

$$dJ = J'_u du + J'_x dx : Adx = (f'_u - A'_u x) du$$

Introducing p leads to

$$A^T p = J'_x \Rightarrow J'_x dx = dx A^T p = (Adx)p = p(f'_u - A'_u x) du$$

$$\text{Therefore } dJ = (J'_u + p(f'_u - A'_u x)) du$$

Consider $u \in \mathbb{R}^m$, $x \in \mathbb{R}^n$, A is $n \times n$ and $p \in \mathbb{R}^n$. p is independent of the choice of the differentiation variable (u here) so it is advantageous when all partial derivatives of J are the goal because they all use the same p .

For m derivatives direct mode is $O(n \times m^2)$, reverse mode is $O(n \times m)$.



Reverse Mode (II)

Builds the Lagrangian by associating to each program line a dual variable p : **each code line is seen as an equality constraint.**

```
float E(u){float x1,x2;x1=(1+u)*log(u);x2=x1+cos(u);E=x1*x2;}
```

$$L = p_1[x_1 - (1 + u) \log(u)] + p_2[x_2 - x_1 - \cos(u)] + E - x_1 x_2.$$

Stationarity of L with respect to all variables

- $\partial_{p_i} L = 0$ gives back the program
- $\partial_{x_i} L = 0$ gives the adjoint state
- $\partial_u L = dE/du$ gives the derivative
- $\frac{\partial L}{\partial x_i} = 0$ must be written in reverse order (x_2, x_1)

$$\frac{\partial L}{\partial x_2} = 0 = p_2 - x_1 \quad \frac{\partial L}{\partial x_1} = 0 = p_1 - p_2 - x_2.$$

This gives p_2 first and then p_1 . Now a theorem says that $E'_u = L'_u$ when $\partial_{x_i} L = \partial_{p_i} L = 0$, so

$$\frac{dE}{du} = \frac{\partial L}{\partial u} = p_2 x_1 \sin(u) - p_1 \left(\log(u) + \frac{1+u}{u} \right)$$



Conclusion

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

Thank you for the invitation

