The Structured Automatic Differentiation Approach for Efficiently Computing Gradients from Monte Carlo Process

By Zhongheng(Leo) Yuan
Supervisor: Prof. Thomas F. Coleman
Waterloo, Ontario, Canada, 2016
Introduction

1. Background
2. Monte Carlo Process for Portfolio Pricing
3. Gradients Computation using Structured Automatic Differentiation (SAD)
4. Numerical Results
1. **Background**

- **Task - Evaluating gradients**
  - E.g. “Greeks” computation in finance
  - Monte Carlo Process
  - Numerically expensive

- **Methods:**
  - Symbolic Computation
  - Finite Difference
  - Automatic Differentiation
1. Background

- Automatic Differentiation
- Gradients Computation
- Monte Carlo Process

Structured Automatic Differentiation
1. Background

- **Automatic Differentiation Background**
  - Forward-mode
    - Chain rule
    - Atomic functions: +,-,\*,/,$\sin,\cos$…
  - Reverse-mode
    - Efficient tool for gradients computation
    - Computational tape
    - “Checkpointing”
1. **Background**

- **Scalar-valued mapping:**

  \[ f : \mathbb{R}^n \to \mathbb{R}^1 \]

- **Computational graph:**

  \[
  \begin{align*}
  \text{Solve for } y_1 : & \quad F_1^E(x, y_1) = \tilde{F}_1(x) - M_1 \cdot y_1 = 0 \\
  \text{Solve for } y_2 : & \quad F_2^E(x, y_1, y_2) = \tilde{F}_2(x, y_1) - M_2 \cdot y_2 = 0 \\
  \vdots \\
  \text{Solve for } y_p : & \quad F_p^E(x, y_1, y_2, \ldots, y_{p-1}, y_p) = \tilde{F}_p(x, y_1, y_2, \ldots, y_{p-1}) - M_p \cdot y_p = 0 \\
  \text{Solve for } z : & \quad F_{p+1}^E(x, y_1, y_2, \ldots, y_{p+1}) = \tilde{f}(x, y_1, y_2, \ldots, y_p) - z = 0
  \end{align*}
  \]
1. Background

- Extended Jacobian:

\[
J^E = \begin{pmatrix}
    J_x^1 & -M_1 & \vdots & \vdots \\
    J_y^1 & J_x^2 & -M_2 & \vdots \\
    \vdots & \vdots & \ddots & \vdots \\
    J_x^p & J_y^1 & J_y^2 & \cdots & -M_p \\
    \nabla \tilde{f}_x & \nabla \tilde{f}_y^T & \nabla \tilde{f}_z^T & \cdots & \nabla \tilde{f}_y^T
\end{pmatrix}
= \begin{pmatrix}
    A \\
    \nabla \tilde{f}_x^T \\
\end{pmatrix}
\begin{pmatrix}
    L \\
    \nabla \tilde{f}_y^T
\end{pmatrix}
\]

- Compute gradients by Schur-complement:

\[
\nabla f^T(x) = \nabla \tilde{f}_x^T - \nabla \tilde{f}_y^T L^{-1} A.
\]
1. **Background**

- **Time and Space**
  - Scalar-valued mapping - $f : \mathbb{R}^n \to \mathbb{R}^1$
  - Time for evaluating $f$ - $\omega(f)$
  - Space for evaluating $f$ - $\sigma(f)$
  - $\sigma(f) \ll \omega(f)$

<table>
<thead>
<tr>
<th></th>
<th>Running time</th>
<th>Space Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward-mode AD</td>
<td>$n^*\omega(f)$</td>
<td>$\sigma(f)$</td>
</tr>
<tr>
<td>Reverse-mode AD</td>
<td>$1^*\omega(f)$</td>
<td>$\omega(f)$</td>
</tr>
</tbody>
</table>
2. Monte Carlo Process for Portfolio Pricing
2. Monte Carlo Process for Portfolio Pricing

Example:

- Paths = $10^4$
- Timesteps = $10^2$
- Cost of each timestep: $\omega(g_{ij} \rightarrow g_{i(j+1)}) = 10^4$

$\rightarrow \omega(f) = 10^4 \times 10^4 \times 10^2 = 10^{10}$

<table>
<thead>
<tr>
<th>Reverse-Mode</th>
<th>Direct AD</th>
<th>Structured AD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space requirement</td>
<td>$10^{10}$</td>
<td>?</td>
</tr>
</tbody>
</table>
2. Monte Carlo Process for Portfolio Pricing

Evaluation of a single path

Evaluation of p paths
3. Gradients Computation using Structured AD

Evaluation of a single path

- **Composite function:**
  - a highly recursive mapping: \( F : \mathbb{R}^n \rightarrow \mathbb{R}^m \)
  - \( F(x) = \tilde{F}(\tilde{F}_T(\tilde{F}_{T-1}(\ldots (\tilde{F}_1(x))\ldots))) \)
  - Simplified Jacobian calculation by its structure:
    \[
    J = \mathcal{J} \cdot \mathcal{J}_T \cdot \mathcal{J}_{T-1} \ldots \cdot \mathcal{J}_1,
    \]
3. Gradients Computation using Structured AD

Evaluation of p Paths

- **GPS Function:**
  - Mapping: \( F : \mathbb{R}^n \to \mathbb{R}^m \)
  - Computational graph:
    - Solve for \( y_i : \tilde{F}_i(x) - y_i = 0, \ i = 1, \ldots, p \)
    - Solve for \( F(x) : \tilde{F}(y_1, y_2, \ldots, y_p) - F(x) = 0 \)
  - Each path is independent
  - Last step is mean operation - simple!
3. Gradients Computation using Structured AD

Evaluation of p Paths

- **GPS Function’s special structure**
  - Extended Jacobian:
    
    \[
    J^E = \begin{pmatrix}
    J^1_x & -I \\
    J^2_x & 0 & -I \\
    \vdots & \vdots & \vdots & \ddots \\
    J^p_x & 0 & \cdots & 0 & -I \\
    0 & \nabla f^T_{y_1} & \nabla f^T_{y_2} & \cdots & \cdots & \nabla f^T_{y_p}
    \end{pmatrix} = \begin{pmatrix} A \\ \nabla f^T_x \\ \nabla f^T_{y_1} \end{pmatrix}.
    \]
  - Gradients computation:
    \[
    \nabla f^T(x) = \nabla f^T_x - \nabla f^T_y L^{-1} A. \quad \Rightarrow
    \]
    \[
    \nabla f^T(x) = - (\nabla f^T_y L^{-1}) A = \nabla f^T_{y_1} \cdot J^1_x + \nabla f^T_{y_2} \cdot J^2_x + \cdots + \nabla f^T_{y_p} \cdot J^p_x
    \]
3. Gradients Computation using Structured AD

Minimizing Memory Requirement

● **Three reverse-mode AD methods:**
  ○ Direct Automatic Differentiation - DAD
  ○ Structured Automatic Differentiation (PathOnly) - SAD-PathOnly
  ○ Structured Automatic Differentiation (Segment) - SAD

● **Fast Memory v.s. Slow Memory**
  ○ Fast memory is limited!
  ○ Affects the theoretical running time
3. Gradients Computation using Structured AD

Example:

- Paths = $10^4$
- Timesteps = $10^2$
- Cost of each timestep: $\omega(g_{ij} \rightarrow g_{i(j+1)}) = 10^4$

$\rightarrow \omega(f) = 10^4 \times 10^4 \times 10^2 = 10^{10}$

<table>
<thead>
<tr>
<th>Reverse-Mode</th>
<th>Direct AD</th>
<th>SAD-PathOnly</th>
<th>SAD (Segment)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space requirement</td>
<td>$10^{10}$</td>
<td>$10^6$</td>
<td>$10^4$</td>
</tr>
</tbody>
</table>
4. Numerical Results

Experiment Design

- **Machine:**
  - RAM: 4G
  - CPU: AMD 2.90Ghz
  - Hard Drive: 512GB
  - Platform: Win7 Pro
  - Software: Matlab2015b
  - Package: AD toolbox ADMAT 2.0
4. Numerical Results

Experiment Design

- **Heston Model:**

\[ \Delta S^i_t = \mu S^i_{t-1} \Delta t + \sqrt{v^i_{t-1}} S^i_{t-1} \Delta W^S_{i-1} \]

\[ \Delta v^i_t = \kappa (\theta - v^i_{t-1}) \Delta t + \zeta \sqrt{v^i_{t-1}} \Delta W^\nu_{i-1} \]

<table>
<thead>
<tr>
<th>Initial Inputs for Heston Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>abs(randn(NAssets,1)) a vector of positive random initial stock prices</td>
</tr>
<tr>
<td>$\mu$</td>
<td>0.005 risk free rate</td>
</tr>
<tr>
<td>$v_0$</td>
<td>0.05 a vector of initial volatility</td>
</tr>
<tr>
<td>$\theta$</td>
<td>0.05 mean-reverting volatility</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>0.1 rate of reverting</td>
</tr>
<tr>
<td>$\zeta$</td>
<td>0.025 volatility of volatility</td>
</tr>
<tr>
<td>$K$</td>
<td>mean($S_0$) strike price as the average of all $S_0$ prices</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0.5 correlation of Wiener Process</td>
</tr>
</tbody>
</table>
4. Numerical Results

Test I - SAD v.s. DAD

- NAssets=100, NSegments=252
4. Numerical Results

Test I - SAD v.s. DAD

- NPaths=400, NSegments=252
4. Numerical Results

Test II - SAD v.s. SAD-PathOnly

- NAssets=10,000, NPaths=10

---

**Memory**

- SAD
- SAD-PathOnly

**Running Time**

- SAD
- SAD-PathOnly
Conclusion

**Structured reverse-mode Automatic Differentiation**

- Powerful tool for gradients computation in Monte Carlo process
- Time proportional to time needed for function evaluation
- Memory requirement significantly reduced
- Fast memory v.s. slow memory

**Nested Monte Carlo Process**

- Similar structure
  - Composite Function
  - Generalized Partially Separable (GPS) Function