Exploiting Intermediate Sparsity in Computing Derivatives for a Leapfrog Scheme

Roland Schäfer

IGPM — RWTH Aachen

2006-08-15
Outline

1. What is Leapfrog

2. Black-Box Approach

3. Intermediate Sparsity (IS) Approach, Compressed Jacobian

4. 2d-Example Shallow Water
   - Simple Update
   - Complex Update

5. Conclusion
1 What is Leapfrog

2 Black-Box Approach

3 Intermediate Sparsity (IS) Approach, Compressed Jacobian

4 2d-Example Shallow Water
   - Simple Update
   - Complex Update

5 Conclusion
What is Leapfrog

Target: Calculate $Z(T)$ from $Z(0)$ (initial value) and $W$ (parameter)
What is Leapfrog

Target: Calculate $Z(T)$ from $Z(0)$ (initial value) and $W$ (parameter)

Leapfrog: $Z(t + 1) = H(Z(t), Z(t - 1), W)$
What is Leapfrog

Target: Calculate $Z(T)$ from $Z(0)$ (initial value) and $W$ (parameter)

Leapfrog: $Z(t + 1) = H(Z(t), Z(t - 1), W)$

Leapfrog Scheme (LS)

Initialize $Z(0)$ and $W$

Compute $Z(1)$

for $t = 1$ to $T - 1$ do

$Z(t + 1) = H(Z(t), Z(t - 1), W)$

end do
1. What is Leapfrog

2. Black-Box Approach

3. Intermediate Sparsity (IS) Approach, Compressed Jacobian

4. 2d-Example Shallow Water
   - Simple Update
   - Complex Update

5. Conclusion
Calculate Derivatives

Let $X$ be a subset of $s$ elements from the $n + p$ sized $[Z(0), W]$. We want:

$$
\frac{dZ(T)}{dX}
$$
Calculate Derivatives

Let $X$ be a subset of $s$ elements from the $n + p$ sized $[Z(0), W]$. We want:

$$\frac{dZ(T)}{dX}$$

Black-Box Approach (BB)

Initialize $[Z(0), \frac{dZ(0)}{dX}]$ and $[W, \frac{dW}{dX}]$

Compute $[Z(1), \frac{dZ(1)}{dX}]$

for $t = 1$ to $T - 1$ do

$$\left[ Z(t + 1), \frac{dZ(t+1)}{dX} \right] = \hat{H} \left( Z(t), \frac{dZ(t)}{dX}, Z(t - 1), \frac{dZ(t-1)}{dX}, W, \frac{dW}{dX} \right)$$

end do
Complexity of BB

Operations:

- Computation of $H$ takes $f_H$ flops
Complexity of BB

Operations:

- Computation of $H$ takes $f_H$ flops
- Computation of $Z(T)$ takes $O(f_H T)$ flops
Complexity of BB

Operations:

- Computation of $H$ takes $f_H$ flops
- Computation of $Z(T)$ takes $O(f_H T)$ flops
- Computation of $[Z(T), \frac{dZ(T)}{dX}]$ takes $O(s \cdot f_H T)$ flops
### Complexity of BB

**Operations:**

- Computation of $H$ takes $f_H$ flops
- Computation of $Z(T)$ takes $O(f_H T)$ flops
- Computation of $[Z(T), \frac{dZ(T)}{dX}]$ takes $O(s \cdot f_H T)$ flops

**Memory:**

- need to save two timesteps
Complexity of BB

Operations:

- Computation of $H$ takes $f_H$ flops
- Computation of $Z(T)$ takes $O(f_H T)$ flops
- Computation of $[Z(T), \frac{dZ(T)}{dX}]$ takes $O(s \cdot f_H T)$ flops

Memory:

- need to save two timesteps
- $Z \in \mathbb{R}^n$, $W \in \mathbb{R}^p$
  therefore computation of $H$ takes $O(2n + p)$ words of storage
Complexity of BB

Operations:

- Computation of $H$ takes $f_H$ flops
- Computation of $Z(T)$ takes $O(f_H T)$ flops
- Computation of $[Z(T), \frac{dZ(T)}{dX}]$ takes $O(s \cdot f_H T)$ flops

Memory:

- need to save two timesteps
- $Z \in \mathbb{R}^n$, $W \in \mathbb{R}^p$
  - therefore computation of $H$ takes $O(2n + p)$ words of storage
- BB takes $O(s \cdot (2n + p))$ words of storage
Outline

1. What is Leapfrog

2. Black-Box Approach

3. Intermediate Sparsity (IS) Approach, Compressed Jacobian

4. 2d-Example Shallow Water
   - Simple Update
   - Complex Update

5. Conclusion
Leapfrog Scheme:

\[ Z(t + 1) = H(Z(t), Z(t - 1), W) \]
Motivation

Leapfrogs Scheme:

\[ Z(t + 1) = H(Z(t), Z(t - 1), W) \]

Differentiate w.r.t. \( X \):

\[
\frac{dZ(t + 1)}{dX} = \frac{\partial H}{\partial Z(t)} \cdot \frac{dZ(t)}{dX} + \frac{\partial H}{\partial Z(t - 1)} \cdot \frac{dZ(t - 1)}{dX} + \frac{\partial H}{\partial W} \cdot \frac{dW}{dX}
\]
Motivation

Leapfrog Scheme:

\[ Z(t + 1) = H(Z(t), Z(t - 1), W) \]

Differentiate w.r.t. \( X \):

\[ \frac{dZ(t + 1)}{dX} = \frac{\partial H}{\partial Z(t)} \cdot \frac{dZ(t)}{dX} + \frac{\partial H}{\partial Z(t - 1)} \cdot \frac{dZ(t - 1)}{dX} + \frac{\partial H}{\partial W} \cdot \frac{dW}{dX} \]

Fact

\( \frac{\partial H}{\partial \ldots} \) is sparse for PDE problems
Motivation

Leapfrog Scheme:

\[ Z(t + 1) = H(Z(t), Z(t - 1), W) \]

Differentiate w.r.t. \( X \):

\[
\frac{dZ(t + 1)}{dX} = \frac{\partial H}{\partial Z(t)} \cdot \frac{dZ(t)}{dX} + \frac{\partial H}{\partial Z(t - 1)} \cdot \frac{dZ(t - 1)}{dX} + \frac{\partial H}{\partial W} \cdot \frac{dW}{dX}
\]

Fact

\( \frac{\partial H}{\partial \ldots} \) is sparse for PDE problems

Exploit this fact with cheap “sparse matrix – matrix” multiplications
Why is the matrix sparse

$H$ typically comes from a stencil, which depends only on a few, neighbored cells.
Why is the matrix sparse

$H$ typically comes from a stencil, which depends only on a few, neighbored cells

In forthcoming example:
max. 13 non-zero entries of $\left[ \frac{\partial H}{\partial Z(\text{t})}, \frac{\partial H}{\partial Z(\text{t-1})}, \frac{\partial H}{\partial W} \right]$

Figure 1. The nonzero structure of the matrix $\left[ \frac{\partial H}{\partial Z(\text{t})}, \frac{\partial H}{\partial Z(\text{t-1})}, \frac{\partial H}{\partial W} \right]$.
Intermediate Sparsity Approach (IS)

Assume: “sparse matrix – matrix” multiplications are cheap
**Intermediate Sparsity Approach (IS)**

Assume: “sparse matrix – matrix” multiplications are cheap

**Intermediate Sparsity Approach (IS)**

Initialize \([Z(0), \frac{dZ(0)}{dX}]\) and \([W, \frac{dW}{dX}]\).

Compute \([Z(1), \frac{dZ(1)}{dX}]\).

for \(t = 1\) to \(T - 1\) do

**Step 1:** Compute \(Z(t + 1)\) and \(\frac{\partial H}{\partial Z(t)}, \frac{\partial H}{\partial Z(t-1)}, \frac{\partial H}{\partial W}\)

**Step 2:** Compute\(^*\) \(\frac{dZ(t+1)}{dX}\) via matrix-matrix multiplication

end do

\(^*\) via \(\frac{dZ(t+1)}{dX} = \frac{\partial H}{\partial Z(t)} \cdot \frac{dZ(t)}{dX} + \frac{\partial H}{\partial Z(t-1)} \cdot \frac{dZ(t-1)}{dX} + \frac{\partial H}{\partial W} \cdot \frac{dW}{dX}\)
Complexity for IS-SL

Assume: “matrix-matrix” multiplications are optimized for sparse matrices: sparse linear algebra (SL)
Assume: “matrix-matrix” multiplications are optimized for sparse matrices: sparse linear algebra (SL)

Operations:

- Stencil size is $O(\kappa)$
Complexity for IS-SL

Assume: “matrix-matrix” multiplications are optimized for sparse matrices: sparse linear algebra (SL)

Operations:

- Stencil size is $O(\kappa)$
- Step 1 needs a total of $O(\kappa f_H T)$ flops (instead of $O(sf_H T)$)
Complexity for IS-SL

Assume: “matrix-matrix” multiplications are optimized for sparse matrices: sparse linear algebra (SL)

Operations:

- Stencil size is $O(\kappa)$
- Step 1 needs a total of $O(\kappa f_H T)$ flops (instead of $O(sf_H T)$)
- matrix-matrix multiplication needs $O(snT)$
Assume: “matrix-matrix” multiplications are optimized for sparse matrices: sparse linear algebra (SL)

Operations:

- Stencil size is \(O(\kappa)\)
- Step 1 needs a total of \(O(\kappa f_H T)\) flops (instead of \(O(sf_H T)\))
- matrix-matrix multiplication needs \(O(snT)\)

Memory:

- Step 1 needs \(O(\kappa(2n + p))\) words of storage
Assume: “matrix-matrix” multiplications are optimized for sparse matrices: sparse linear algebra (SL)

Operations:

- Stencil size is $O(\kappa)$
- Step 1 needs a total of $O(\kappa f_{HT})$ flops (instead of $O(sf_{HT})$)
- matrix-matrix multiplication needs $O(snT)$

Memory:

- Step 1 needs $O(\kappa(2n + p))$ words of storage
- Step 2 needs $O(sn)$ words of storage
Another way: Compressed Jacobians (IS-CJ)

Let $S^1$, $S^2$ be suitable chosen seed matrices with $\lambda_1$, $\lambda_2$ columns for $\frac{dZ(t)}{dX}$ and $\frac{dZ(t-1)}{dX}$
Another way: Compressed Jacobians (IS-CJ)

Let $S^1$, $S^2$ be suitable chosen seed matrices with $\lambda_1, \lambda_2$ columns for $\frac{dZ(t)}{dX}$
and $\frac{dZ(t-1)}{dX}$

Obtain a compressed version of $\frac{\partial H}{\partial \ldots}$
Another way: Compressed Jacobians (IS-CJ)

Let $S^1, S^2$ be suitable chosen seed matrices with $\lambda_1, \lambda_2$ columns for $\frac{dZ(t)}{dX}$ and $\frac{dZ(t-1)}{dX}$

Obtain a compressed version of $\frac{\partial H}{\partial...}$

Complexity: $(\lambda = \lambda_1 + \lambda_2 + p)$

- Computation: $O(\lambda f_H T) + O(snT)$
- Memory: $O(\lambda(2n + p)) + O(sn)$
### Complexity overview

<table>
<thead>
<tr>
<th></th>
<th>BB</th>
<th>IS–SL</th>
<th>IS–CJ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Computation</strong></td>
<td>$O(sf_H T)$</td>
<td>$O(\kappa f_H T) + O(snT)$</td>
<td>$O(\lambda f_H T) + O(snT)$</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>$O(s(2n + p))$</td>
<td>$O(\kappa(2n + p)) + O(sn)$</td>
<td>$O(\lambda(2n + p)) + O(sn)$</td>
</tr>
</tbody>
</table>


## Complexity overview

<table>
<thead>
<tr>
<th></th>
<th>BB</th>
<th>IS–SL</th>
<th>IS–CJ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Computation</strong></td>
<td>$O(sf_HT)$</td>
<td>$O(\kappa f_HT) + O(snT)$</td>
<td>$O(\lambda f_HT) + O(snT)$</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>$O(s(2n + p))$</td>
<td>$O(\kappa(2n + p)) + O(sn)$</td>
<td>$O(\lambda(2n + p)) + O(sn)$</td>
</tr>
</tbody>
</table>

When is IS faster then BB?
Complexity overview

<table>
<thead>
<tr>
<th></th>
<th>BB</th>
<th>IS–SL</th>
<th>IS–CJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>$O(sf_H T)$</td>
<td>$O(\kappa f_H T) + O(snT)$</td>
<td>$O(\lambda f_H T) + O(snT)$</td>
</tr>
<tr>
<td>Storage</td>
<td>$O(s(2n + p))$</td>
<td>$O(\kappa(2n + p)) + O(sn)$</td>
<td>$O(\lambda(2n + p)) + O(sn)$</td>
</tr>
</tbody>
</table>

When is IS faster then BB?

Assume: $\kappa, \lambda \ll s$
### Complexity overview

<table>
<thead>
<tr>
<th></th>
<th>BB</th>
<th>IS–SL</th>
<th>IS–CJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>$O(sf_H T)$</td>
<td>$O(\kappa f_H T) + O(snT)$</td>
<td>$O(\lambda f_H T) + O(snT)$</td>
</tr>
<tr>
<td>Storage</td>
<td>$O(s(2n + p))$</td>
<td>$O(\kappa(2n + p)) + O(sn)$</td>
<td>$O(\lambda(2n + p)) + O(sn)$</td>
</tr>
</tbody>
</table>

When is IS faster then BB?

**Assume:** $\kappa, \lambda \ll s$

IS is faster then BB if $O(\kappa f_H T) \gg O(snT)$
Outline

1. What is Leapfrog
2. Black-Box Approach
3. Intermediate Sparsity (IS) Approach, Compressed Jacobian
4. 2d-Example Shallow Water
   - Simple Update
   - Complex Update
5. Conclusion
Example Shallow Water in 2d

Shallow Water used to simulate water flow, where vertical dimension is much smaller than horizontal (shallow).

E.g. rivers, lakes, coastal flow

Variables: water height, $x$-momentum, $y$-momentum (2d)
Example Shallow Water in 2d

Shallow Water used to simulate water flow, where vertical dimension is much smaller than horizontal (shallow).

E.g. rivers, lakes, coastal flow

Variables: water height, x-momentum, y-momentum (2d)

We calculate \( s = n + p \) derivatives

<table>
<thead>
<tr>
<th>Grid size</th>
<th>( n )</th>
<th>( p )</th>
<th>( s = n + p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 \times 11</td>
<td>3 \times 11 \times 11 = 363</td>
<td>4</td>
<td>367</td>
</tr>
<tr>
<td>16 \times 16</td>
<td>3 \times 16 \times 16 = 768</td>
<td>4</td>
<td>772</td>
</tr>
<tr>
<td>21 \times 21</td>
<td>3 \times 21 \times 21 = 1323</td>
<td>4</td>
<td>1327</td>
</tr>
</tbody>
</table>
Figure 3. Nonzero entries in derivative vectors over time (BB vs IS approach).

We compute $s = n + p$ derivatives with respect to $Z(0)$ and $W$ using the seeding suggested in (3) for $T = 60$ time steps.

In Step 1 of the IS–SL approach we use SparsLinC to compute the sparse Jacobian shown in figure 1. In the IS–CJ approach, we use SparsLinC only in the first time step to determine the sparsity structure and then follow the compressed Jacobian approach in subsequent time steps. This approach is feasible here because the sparsity pattern does not change. Notice that continuous use of SparsLinC could accommodate varying sparsity patterns.

Table 3 contains a summary of the memory requirements of the IS–SL, IS–CJ, and BB approaches. We see that the IS schemes require less memory, since data internal to $H$ require shorter gradients.

### Table 3. Memory requirements in mbytes.

<table>
<thead>
<tr>
<th>Grid size</th>
<th>IS–SL</th>
<th>IS–CJ</th>
<th>BB</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 × 11</td>
<td>3.72</td>
<td>3.85</td>
<td>4.70</td>
</tr>
<tr>
<td>16 × 16</td>
<td>13.61</td>
<td>13.84</td>
<td>18.82</td>
</tr>
<tr>
<td>21 × 21</td>
<td>37.82</td>
<td>38.16</td>
<td>53.31</td>
</tr>
</tbody>
</table>

Memory requirements in megabytes
## & Runtime

<table>
<thead>
<tr>
<th>Grid size (platform)</th>
<th>SL</th>
<th>CJ</th>
<th>MM</th>
<th>IS–SL</th>
<th>IS–CJ</th>
<th>BB</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 × 11 (IBM)</td>
<td>4.90</td>
<td>1.93</td>
<td>8.03</td>
<td>12.93</td>
<td>9.96</td>
<td>4.24</td>
</tr>
<tr>
<td>16 × 16 (IBM)</td>
<td>17.77</td>
<td>8.70</td>
<td>38.66</td>
<td>56.43</td>
<td>47.36</td>
<td>36.68</td>
</tr>
<tr>
<td>21 × 21 (IBM)</td>
<td>42.98</td>
<td>21.51</td>
<td>119.32</td>
<td>162.30</td>
<td>140.83</td>
<td>71.98</td>
</tr>
</tbody>
</table>

**Runtime in seconds**

- **SL**: calculate sparse Jacobians
- **CJ**: calculate compressed Jacobians
- **MM**: matrix-matrix multiplications
- **SL+MM = IS-SL, CJ+MM = IS-CJ**
Complexity overview

<table>
<thead>
<tr>
<th></th>
<th>BB</th>
<th>IS–SL</th>
<th>IS–CJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>( O(s f_H T) )</td>
<td>( O(\kappa f_H T) + O(snT) )</td>
<td>( O(\lambda f_H T) + O(snT) )</td>
</tr>
<tr>
<td>Storage</td>
<td>( O(s(2n + p)) )</td>
<td>( O(\kappa(2n + p)) + O(sn) )</td>
<td>( O(\lambda(2n + p)) + O(sn) )</td>
</tr>
</tbody>
</table>

When is IS faster then BB?

Assume: \( \kappa, \lambda \ll s \)

IS is faster then BB if \( O(\kappa f_H T) \gg O(snT) \)
Expensive Update

Lets evaluate $H$ not only one time, but up to 16 times to emulate a function $H$, which is more expensive to calculate.
Expensive Update

Let's evaluate $H$ not only one time, but up to 16 times to emulate a function $H$, which is more expensive to calculate (IS-SL).

![Graph showing speedup vs repetitions for different systems.

Figure 4. Serial speedup of the IS–SL approach (top) and IS–CJ approach (bottom) over the BB approach. The resulting computational behavior of this second strategy is shown in figures 4 and 5. Speedup here is the ratio of CPU time of the BB approach versus the CPU time of the IS approach. Figure 4 shows that we can in fact obtain considerable speedup if the computational weight of the time step update is sufficiently large in comparison with the cost of the matrix-matrix accumulation step. This is also evident in figure 5, which shows the steadily decreasing influence of the matrix-matrix multiply time as the amount of work per time step increases.

6. A parameter identification scenario

There is another scenario where the intermediate sparsity approach is advantageous. Consider the case where one is interested in finding the system parameters, $W \in \mathbb{R}^p$, from
Figure 5. Percentage of time spent in matrix-matrix multiplications for the IS–SL approach (top) and IS–CJ approach (bottom).
Outline

1. What is Leapfrog

2. Black-Box Approach

3. Intermediate Sparsity (IS) Approach, Compressed Jacobian

4. 2d-Example Shallow Water
   - Simple Update
   - Complex Update

5. Conclusion
Exploiting Sparsity works, if $O(\kappa f_H T) \gg O(snT)$
Exploiting Sparsity works, if $O(\kappa f_H T) \gg O(snT)$

this means: evaluation of $H$ must be expensive
Exploiting Sparsity works, if $O(\kappa f_H T) \gg O(snT)$

this means: evaluation of $H$ must be expensive

for example: High-Order code
Exploiting Sparsity works, if \( O(\kappa f_H T) \gg O(snT) \)

this means: evaluation of \( H \) must be expensive

for example: High-Order code

substantial speedup
Conclusion

- Exploiting Sparsity works, if $O(\kappa f_H T) \gg O(snT)$
- this means: evaluation of $H$ must be expensive
- for example: High-Order code
- substantial speedup