Adjoints via Algorithmic Differentiation - an intro

Patrick Heimbach

The University of Texas at Austin

Oden Institute for Computational Engineering and Sciences
Jackson School of Geosciences
Institute for Geophysics

ECCO Summer School 2019
Friday Harbor Laboratories
May 20 to 31, 2019

Why gradients?

- Optimization (inversion, least-squares estimation)
- Comprenensive sensitivity analysis
- Uncertainty characterization and quantification
- Non-normal transient amplification/growth (singular vectors)

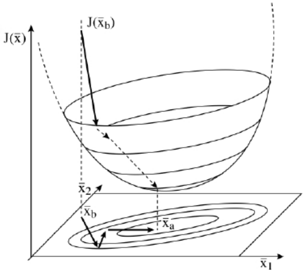
Why gradients? Optimization!

- **Given:**
  - a set of (possibly different types of) observations
  - a numerical model & set of initial/boundary conditions, parameters
- **Question:** (estimation / optimal control problem)
  Find "*optimal*" model trajectory consistent with available observations
- **Approach:** seek minimum of least square cost function

$$\min_{\vec{u}} \{\mathcal{J}(\vec{u})\} = \min_{\vec{u}} \left\{ \sum_i [\,\text{model}_i(\vec{u}) - \text{data}_i\,]^2 \right\}$$

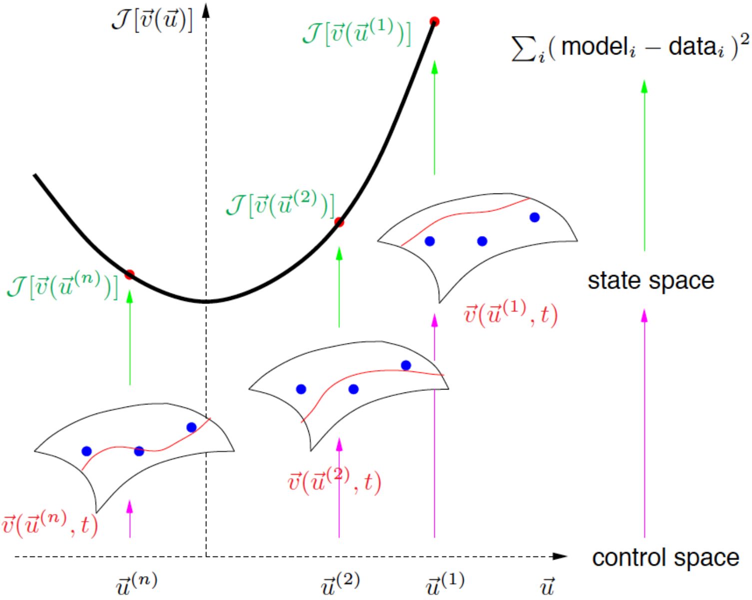
$\longrightarrow$ seek $\boxed{\vec{\nabla}_u \mathcal{J}(\vec{u})}$ to infer update $\Delta\vec{u}$ from variation

$$\vec{u}^{\,n+1} = \vec{u}^{\,n} + \Delta\vec{u}$$



- **Results:** see ECCO
  - optimal/consistent ocean state estimate
  - adjusted initial/boundary value estimates

Why gradients? Optimization!



Why gradients? Sensitivities!

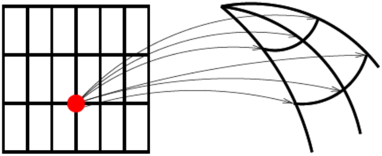
- **Finite difference approach:**
  - Take "guessed" anomaly (e.g. **SST**) and determine its impact on model output (**ice export**)
  - Perturb each input element (**SST**(i,j)) to determine its impact on output (**ice export**).

Impact of ***one input*** on ***all outputs***

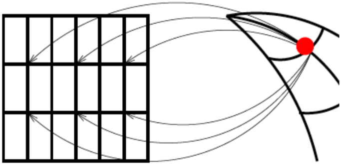
- **Reverse/adjoint approach:**
  - Calculates "full" sensitivity field $\frac{\partial\,\textbf{ice export}}{\partial\,\textbf{SST}(x,y,t)}$
  - Approach: Let $\mathcal{J} =$ **export**, $\vec{u} =$ **SST**(i,j)

$$\longrightarrow \boxed{\vec{\nabla}_u \mathcal{J}(\vec{u})} = \frac{\partial\,\textbf{ice export}}{\partial\,\textbf{SST}(x,y,t)}$$

Sensitivity of ***one output*** to ***all inputs***



forward or finite difference approach



adjoint approach

Why gradients? Uncertainties!

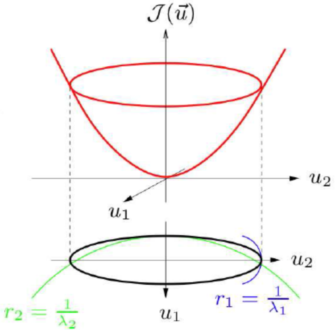
- Consider linear approx. of cost function

$$\mathcal{J}(\vec{u}) = \frac{1}{2}\left(\mathcal{M}(\vec{u}) - \vec{d}\right)^T W \left(\mathcal{M}(\vec{u}) - \vec{d}\right)^T$$
$$\approx \frac{1}{2}(\vec{u} - \vec{u}_0)^T \left(\frac{\partial \mathcal{M}}{\partial u}\right)^T W \left(\frac{\partial \mathcal{M}}{\partial u}\right)(\vec{u} - i$$

- Compare to multivariate Gaussian distribution

$$\mathcal{N}(\vec{u}_0, \Sigma) \propto \exp\left[(\vec{u} - \vec{u}_0)^T \Sigma^{-1} (\vec{u} - \vec{u}_0)\right]$$

- posterior error covariance matrix Σ is inverse of Hessian H of $\mathcal{J}(\vec{u})$ at minimum:



$$H = d_u^2 \mathcal{J}(\vec{u}_{opt})$$
$$= \left(\frac{\partial \mathcal{M}}{\partial u}\right)^T W \left(\frac{\partial \mathcal{M}}{\partial u}\right) + \left(\frac{\partial^2 \mathcal{M}_k}{\partial u_i \partial u_j}\right) W \left(\mathcal{M}(\vec{u}) - \vec{d}\right)$$

- Eigenvalues of H: principal curvatures

$$\frac{\text{largest EV}}{\text{smallest EV}} = \text{conditioning number}$$

- r_i: principal curvatures
- $\det(H^{-1})$: Gauss curvature
- $\mathrm{trace}\,(H^{-1})$: mean curvature

Why gradients? Non-normal Transient Amplification!

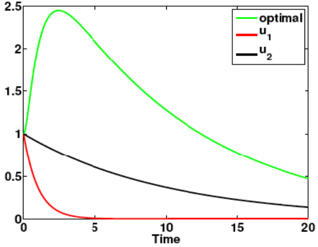
- Consider stable linear system

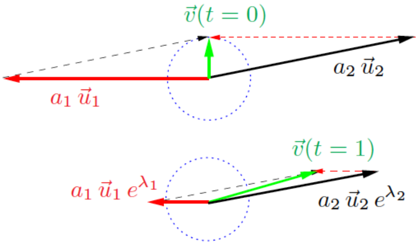
$$\frac{\mathrm{d}\,\vec{v}(t)}{\mathrm{d}\,t} = M\,\vec{v}(t), \quad \vec{v}(t) \to 0 \text{ for } t \to \infty$$

- If M is non-normal, $M \cdot M^T \neq M^T \cdot M$, non-orthogonal eigenvectors:

$$\vec{v}(t) = a_1\,\vec{u_1}\,e^{\lambda_1 t} + a_2\,\vec{u_2}\,e^{\lambda_2 t}$$

- If decay timescales very different, i.e. $\lambda_1 << \lambda_2 < 0$, then
  - $a_1\,\vec{u_1}\,e^{\lambda_1 t}$ decays quickly,removing partial cancelation of EV's
  - causing transient amplification for $t \approx 1$
  - leaving mostly $\vec{v}(t) \approx a_2\,\vec{u_2}\,e^{\lambda_2 t} \to 0$ for $t \to \infty$.





A simple example

Introduction – a simple example

Consider model, L, mapping 2-dim. vector $\mathbf{x}$ to $\mathbf{y}$:

Model

$$\mathbf{y} = L(\mathbf{x}) = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 & a \\ -b & 0 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a\,x_2 \\ -b\,x_1 \end{bmatrix} \tag{1}$$

Now, assume observations $[d_1 \quad d_2]^T$ are available for the two elements $[y_1 \quad y_2]^T$, and we can write a misfit or cost function:

Least-squares cost/objective function

$$\begin{aligned} J_0 = J_0(\mathbf{y}) &= \frac{1}{\sigma_1^2}(y_1 - d_1)^2 + \frac{1}{\sigma_2^2}(y_2 - d_2)^2 \\ &= \frac{1}{\sigma_1^2}(ax_2 - d_1)^2 + \frac{1}{\sigma_2^2}(-bx_1 - d_2)^2 \end{aligned} \tag{2}$$

with σ_1, σ_2 prior errors (special case of inverse error covariance).

Introduction – a simple example

Can view J_0 as a *composite* mapping

$$J_0 = J_0(\mathbf{y}) = J_0(L(\mathbf{x})),$$

such that

$$\begin{array}{rcccccc} J_0 : & \mathbf{x} & \longmapsto & \mathbf{y} & \longmapsto & J_0[\mathbf{y}] \\ & \mathbf{x} & \longmapsto & L[\mathbf{x}] & \longmapsto & J_0[L[\mathbf{x}]] \\ & I\!R^m & \longmapsto & I\!R^n & \longmapsto & I\!R \end{array} \qquad (3)$$

- Find the gradient of J_0 with respect to the input variable $\mathbf{x}$.
- Note that, alternatively, or in addition, we could also be interested in the gradient of J_0 with respect to the model parameters $\mathbf{p} = [a \quad b]^T$ – will come back to later.

Of course, the example chosen is very simple, and from eqn. (2) we can readily write down the gradient:

The gradient (with respect to x)

$$\nabla_x J_0^T = \begin{bmatrix} \frac{\partial J_0}{\partial x_1} \\ \frac{\partial J_0}{\partial x_2} \end{bmatrix} = \begin{bmatrix} -\frac{2b}{\sigma_2^2}\left(-bx_1 - d_2\right) \\ \frac{2a}{\sigma_1^2}\left(ax_2 - d_1\right) \end{bmatrix} \tag{4}$$

DONE!

Introduction – some terminology

- *dependent versus independent variables:*
  - J_0 (or L): dependent variable whose gradient is sought (cost/objective function; target quantity of interest – QoI) often scalar-valued!
  - $\vec{u}$ or $\mathbf{x}(0)$: independent or control variables variables with respect to which the dependent variable is differentiated
- *forward / reverse mode:*
  - tangent linear model: forward mode
  - adjoint model: reverse mode
- *active, passive, required variables:*
  e.g., for $J_0 = a^2 x_2^2 + x_1^2$:
  - active: x_1, x_2 (variables that are subject to differentiation)
  - passive: a (variables NOT subject to differentiation)
  - required: x_1, x_2 (variables needed to evaluate derivative)

Introduction – a simple example

The conventional approach: directional derivative

$$\frac{\partial J_0}{\partial x_i} = \frac{J_0(\mathbf{x} + \epsilon\,\mathbf{e}_i) - J_0(\mathbf{x})}{\epsilon}$$

for small ϵ, and required for each direction $\mathbf{e}_i$

$$\mathbf{e}_1 = [1 \quad 0]^T, \qquad \mathbf{e}_2 = [0 \quad 1]^T$$

Serveral shortcomings:

- If the dimension of $\mathbf{x}$ was very large (e.g. 10^7 instead of 2) and calculation of J_0 expensive, performing 10^7 perturbation calculations would be prohibitive;
- Accuracy depends on coice of ϵ and finite-differencing scheme used (here we used the simplest possible)

Introduction – a simple example

Consider how perturbations $\delta\mathbf{x}$ in $\mathbf{x}$ are mapped to perturbations $\delta\mathbf{y}$ in $\mathbf{y} = L\mathbf{x}$. We define the linearized model dL via

$$\delta\mathbf{y} = dL\,\delta\mathbf{x}$$

$$\begin{aligned}
\begin{bmatrix} \delta x_1 \\ \delta x_2 \end{bmatrix} \longmapsto \begin{bmatrix} \delta y_1 \\ \delta y_2 \end{bmatrix} &= \begin{bmatrix} \frac{\partial y_1}{\partial x_1}\delta x_1 + \frac{\partial y_1}{\partial x_2}\delta x_2 \\ \frac{\partial y_2}{\partial x_1}\delta x_1 + \frac{\partial y_2}{\partial x_2}\delta x_2 \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix} \cdot \begin{bmatrix} \delta x_1 \\ \delta x_2 \end{bmatrix} \\
&= \begin{bmatrix} 0 & a \\ -b & 0 \end{bmatrix} \cdot \begin{bmatrix} \delta x_1 \\ \delta x_2 \end{bmatrix} = \begin{bmatrix} a\,\delta x_1 \\ -b\,\delta x_2 \end{bmatrix}
\end{aligned} \tag{5}$$

N.B.: Since L is linear, the Jacobian dL is identical to L (a choice to simplify our calculation for now).

Introduction – a simple example

Now, consider the total variation of J_0 with respect to $\mathbf{y}$:

$$\delta J_0 = \frac{\partial J_0}{\partial y_1}\delta y_1 + \frac{\partial J_0}{\partial y_2}\delta y_2 = \left\langle \frac{\partial J_0}{\partial \mathbf{y}}^T, \delta \mathbf{y} \right\rangle \tag{6}$$

with general inner product $< . , . >$.

We can obtain gradient using formal definition
$< A^T x, y > = < x, Ay >$ of the adjoint:

$$\begin{aligned} \delta J_0 &= \left\langle \frac{\partial J_0}{\partial \mathbf{y}}^T, \delta \mathbf{y} \right\rangle \\ &= \left\langle \frac{\partial J_0}{\partial \mathbf{y}}^T, dL\,\delta \mathbf{x} \right\rangle = \left\langle dL^T \frac{\partial J_0}{\partial \mathbf{y}}^T, \delta \mathbf{x} \right\rangle = \left\langle \frac{\partial J_0}{\partial \mathbf{x}}^T, \delta \mathbf{x} \right\rangle \end{aligned} \tag{7}$$

We obtain general expressions for the tangent linear model and its dual, the adjoint model:

$$
\begin{array}{rcccccc}
dJ_0 : & \delta\mathbf{x} & \longrightarrow & \delta\mathbf{y} = dL \cdot \delta\mathbf{x} & \longrightarrow & \delta J_0 = \nabla_y J_0 \cdot \delta\mathbf{y} \\
d^* J_0 : & \delta^*\mathbf{x} = dL^T \cdot \delta^*\mathbf{y} & \longleftarrow & \delta^*\mathbf{y} = \nabla_y J_0^T & \longleftarrow & \delta^* J_0 = 1
\end{array}
\tag{8}
$$

with

$$
\delta^*\mathbf{x} \; = \; \nabla_x J_0^T \; = \; \frac{\partial \mathbf{y}}{\partial \mathbf{x}}^T \cdot \frac{\partial J_0}{\partial \mathbf{y}}^T \cdot \delta J_0^T \tag{9}
$$

For our example, we obtain:

$$
\begin{aligned}
\delta J_0 &= \frac{2}{\sigma_1^2}(y_1 - d_1)\,\delta y_1 \;+\; \frac{2}{\sigma_2^2}(y_2 - d_2)\,\delta y_2 \\
&= \left[\frac{2}{\sigma_1^2}(y_1 - d_1) \quad \frac{2}{\sigma_2^2}(y_2 - d_2) \right] \cdot \begin{bmatrix} \delta y_1 \\ \delta y_2 \end{bmatrix} \\
&= \left[\frac{2}{\sigma_1^2}(a x_2 - d_1) \quad \frac{2}{\sigma_2^2}(-b x_1 - d_2) \right] \cdot \begin{bmatrix} 0 & a \\ -b & 0 \end{bmatrix} \cdot \begin{bmatrix} \delta x_1 \\ \delta x_2 \end{bmatrix} \\
&= \left[-\frac{2b}{\sigma_2^2}(-b x_1 - d_2) \quad \frac{2a}{\sigma_1^2}(a x_2 - d_1) \right] \cdot \begin{bmatrix} \delta x_1 \\ \delta x_2 \end{bmatrix}
\end{aligned}
\tag{10}
$$

Introduction – a simple example

$$
\begin{aligned}
\delta^*\mathbf{x} = \begin{bmatrix} \delta^* x_1 \\ \delta^* x_2 \end{bmatrix} &= \begin{bmatrix} -\frac{2b}{\sigma_2^2}(-bx_1 - d_2) \\ \frac{2a}{\sigma_1^2}(ax_2 - d_1) \end{bmatrix} \\
&= \begin{bmatrix} 0 & -b \\ a & 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{2}{\sigma_1^2}(ax_2 - d_1) \\ \frac{2}{\sigma_2^2}(-bx_1 - d_2) \end{bmatrix} \cdot \delta^* J_0 \qquad (11) \\
&= dL^T \cdot \delta^*\mathbf{y} \cdot \delta^* J_0
\end{aligned}
$$

with $\delta^* J_0 = 1$

Introduction: change of control space – same model, different adjoint!

"*THE*" adjoint model?

Introduction: change of control space – same model, different adjoint!

Consider sensitivity of J_0, not with respect to state $\mathbf{x}$, but with respect to model parameters $\mathbf{p} = [a \quad b]^T$.

Direct differentiation yields:

$$\nabla_p J_0^T = \begin{bmatrix} \frac{\partial J_0}{\partial a} \\ \frac{\partial J_0}{\partial b} \end{bmatrix} = \begin{bmatrix} \frac{2}{\sigma_1^2}\left(a x_2 - d_1\right) x_2 \\ -\frac{2}{\sigma_2^2}\left(-b x_1 - d_2\right) x_1 \end{bmatrix}$$

and:

$$\begin{aligned} \delta J_0 &= \frac{\partial J_0}{\partial a}\delta a + \frac{\partial J_0}{\partial b}\delta b \\ &= \begin{bmatrix} \frac{2}{\sigma_1^2}\left(a x_2 - d_1\right) & -\frac{2}{\sigma_2^2}\left(-b x_1 - d_2\right) \end{bmatrix} \cdot \begin{bmatrix} x_2 & 0 \\ 0 & -x_1 \end{bmatrix} \cdot \begin{bmatrix} \delta a \\ \delta b \end{bmatrix} \end{aligned}$$

Introduction: change of control space – same model, different adjoint!

We can readily deduce:

$$\delta^*\mathbf{p} = \begin{bmatrix} \delta^* a \\ \delta^* b \end{bmatrix} = \begin{bmatrix} \frac{2}{\sigma_1^2}(ax_2 - d_1)x_2 \\ -\frac{2}{\sigma_2^2}(-bx_1 - d_1)x_1 \end{bmatrix}$$
$$= \begin{bmatrix} x_2 & 0 \\ 0 & -x_1 \end{bmatrix} \cdot \begin{bmatrix} \frac{2}{\sigma_1^2}(ax_2 - d_1) \\ \frac{2}{\sigma_2^2}(-bx_1 - d_2) \end{bmatrix} \cdot \delta^* J_0 \qquad (12)$$
$$= d\tilde{L}^T \cdot \delta^*\mathbf{y} \cdot \delta^* J_0$$

with corresponding mapping relationship:

$$\begin{array}{lcccccc} dJ_0 : & \delta\mathbf{p} & \longrightarrow & \delta\mathbf{y}(\mathbf{p}) = d\tilde{L} \cdot \delta\mathbf{p} & \longrightarrow & \delta J_0 = \nabla_y J_0 \cdot \delta\mathbf{y} \\ d^* J_0 : & \delta^*\mathbf{p} = d\tilde{L}^T \cdot \delta^*\mathbf{y} & \longleftarrow & \delta^*\mathbf{y} = \nabla_y J_0^T & \longleftarrow & \delta^* J_0 = 1 \end{array} \qquad (13$$

Introduction: preliminary lessons

- There isn't such a thing as *"the adjoint model"*!
  Its form depends crucially on the control problem formulated.
- A strengths of *algorithmic differentiation* is the fact that it can deal much more flexibly with changes to the formulation.
- It isn't even clear what is meant by *"the adjoint model"*:
  - mathematicians refer to the entire expression $\boxed{dL^T \cdot \delta^* \mathbf{y} \cdot \delta^* J_0}$ as the adjoint of the mapping $J_0(L(\mathbf{x}))$,
  - physicists think of L as "the model", dL as "the Jacobian", and $\boxed{dL^T}$ only as "the adjoint";
- The expressions for $\delta^* \mathbf{y} = \nabla_y J_0^T$ remain the same, and it is really dL vs. $d\tilde{L}$ (and their transpose) which change the overall TLM and ADM.

Introduction: can also compute the "joint gradient"

Homework

The time-varying problem

The time-varying problem

Consider time-evolving model:

$$\mathbf{x}(t) \;-\; L\left[\mathbf{x}(t-1)\right] \;=\; 0 \tag{14}$$

Define objective function:
time-mean volume over last $n+1$ timesteps $t_f - n, \ldots, t_f - 1, t_f$:

$$J_0[\mathbf{x}] \;=\; \frac{1}{n+1}\Big(V[\mathbf{x}(t_f - n)] + \ldots + V[\mathbf{x}(t_f)] \Big) \tag{15}$$

Define Lagrangian:

$$J \;=\; J_0[\mathbf{x}] \;-\; \sum_{1}^{t_f} \boldsymbol{\mu}^T(t) \Big\{ \mathbf{x}(t) \;-\; L\left[\mathbf{x}(t-1)\right] \Big\} \tag{16}$$

The time-varying problem: set of normal equations

$$\frac{\partial J}{\partial \boldsymbol{\mu}(t)} = \mathbf{x}(t) - L\left[\mathbf{x}(t-1)\right] = 0 \qquad 1 \le t \le t_f \qquad (17a)$$

$$\frac{\partial J}{\partial \mathbf{x}(t)} = \frac{\partial J_0}{\partial \mathbf{x}(t)} - \boldsymbol{\mu}(t) + \left[\frac{\partial L[\mathbf{x}(t)]}{\partial \mathbf{x}(t)}\right]^T \boldsymbol{\mu}(t+1) = 0 \qquad 0 < t < t_f \qquad (17b)$$

$$\frac{\partial J}{\partial \mathbf{x}(t_f)} = \frac{\partial J_0}{\partial \mathbf{x}(t_f)} - \boldsymbol{\mu}(t_f) = 0 \qquad t = t_f \qquad (17c)$$

$$\frac{\partial J}{\partial \mathbf{x}(0)} = \frac{\partial J_0}{\partial \mathbf{x}(0)} - \left[\frac{\partial L[\mathbf{x}(0)]}{\partial \mathbf{x}(0)}\right]^T \boldsymbol{\mu}(1) \qquad t_0 = 0 \qquad (17d)$$

The time-varying problem: adjoint time-stepping

Successive evaluation backward in time, starting at $t = t_f$:

$$\mu(t_f) = \frac{\partial J_0}{\partial \mathbf{x}(t_f)} = \frac{1}{n+1} \frac{\partial V[\mathbf{x}(t_f)]}{\partial \mathbf{x}(t_f)}$$

$n+1$ time steps earlier, at $t = t_f - n$, and using the results of $\mu(t_f), \ldots, \mu(t_f - n + 1)$, we obtain:

$$\begin{aligned}
\mu(t_f - n) = \frac{1}{n+1} \Bigg\{ & \frac{\partial V[\mathbf{x}(t_f - n)]}{\partial \mathbf{x}(t_f - n)} \\
& + \left[\frac{\partial L[\mathbf{x}(t_f - n)]}{\partial \mathbf{x}(t_f - n)} \right]^T \cdot \frac{\partial V[\mathbf{x}(t_f - n + 1)]}{\partial \mathbf{x}(t_f - n + 1)} \\
& + \ldots \\
& + \left[\frac{\partial L[\mathbf{x}(t_f - n)]}{\partial \mathbf{x}(t_f - n)} \right]^T \cdot \ldots \cdot \left[\frac{\partial L[\mathbf{x}(t_f - 1)]}{\partial \mathbf{x}(t_f - 1)} \right]^T \\
& \cdot \frac{\partial V[\mathbf{x}(t_f)]}{\partial \mathbf{x}(t_f)} \Bigg\}
\end{aligned} \tag{18}$$

The time-varying problem: interpretation

- Lagrange multiplier $\mu(t)$ provides complete sensitivity of J_0 at time t by accumulating all partial derivatives of J_0 with respect to $\mathbf{x}$ from each time step $t_f, t_f - 1, \ldots, t$
- Those partials taken at later times $t+1, \ldots, t_f$, are propagated to time t via the adjoint model (ADM), which is the transpose $\left[\frac{\partial L[\mathbf{x}(t)]}{\mathbf{x}(t)}\right]^T$ of the model Jacobian or tangent linear model (TLM), $\frac{\partial \mathbf{x}(t+1)}{\mathbf{x}(t)} = \frac{\partial L[\mathbf{x}(t)]}{\mathbf{x}(t)}$
- contributions from different times linearly superimposed
- Simplifying the example objective function: instead of the time-mean, only the volume at the last time step t_f is chosen: now all terms except the one containing $\frac{\partial V[\mathbf{x}(t_f)]}{\partial \mathbf{x}(t_f)}$ vanish

The time-varying problem: the chain rule

$$
\begin{array}{rcccc}
J_0 : & \mathbf{x}(0) & \longmapsto & \mathbf{y} = \mathbf{x}(t_f) & \longmapsto & J_0[\mathbf{y}] \\
 & \mathbf{x}(0) & \longmapsto & L[\mathbf{x}(t_f - 1)] & \longmapsto & V[L[\mathbf{x}(t_f - 1)]]
\end{array}
$$

is composite mapping for special case, $J_0 = V[\mathbf{x}(t_f)] = V[\mathbf{y}]$

$$
\begin{aligned}
J_0 &= V[\mathbf{x}(t_f)] \\
&= V[L[L[\ldots L[\mathbf{x}(0)]]]]
\end{aligned}
$$

and corresponding perturbation:

$$
\begin{aligned}
\delta J_0 &= \frac{\partial V}{\partial \mathbf{x}(t_f)} \delta \mathbf{x}(t_f) \\
&= \frac{\partial V}{\partial \mathbf{x}(t_f)} \cdot \frac{\partial \mathbf{x}(t_f)}{\partial \mathbf{x}(t_f - 1)} \cdot \ldots \cdot \frac{\partial \mathbf{x}(1)}{\partial \mathbf{x}(0)} \cdot \delta \mathbf{x}(0)
\end{aligned}
$$

The time-varying problem: the chain rule

$$
\begin{aligned}
\delta J_0 &= \left\langle \frac{\partial V}{\partial \mathbf{y}} \;\middle|\; \delta \mathbf{y} \right\rangle \\
&= \left\langle \frac{\partial V}{\partial \mathbf{x}(t_f)} \;\middle|\; \frac{\partial \mathbf{x}(t_f)}{\partial \mathbf{x}(t_f - 1)} \cdot \ldots \cdot \frac{\partial \mathbf{x}(1)}{\partial \mathbf{x}(0)} \cdot \delta \mathbf{x}(0) \right\rangle \\
&= \left\langle \left[\frac{\partial \mathbf{x}(1)}{\partial \mathbf{x}(0)} \right]^T \cdot \ldots \cdot \left[\frac{\partial \mathbf{x}(t_f)}{\partial \mathbf{x}(t_f - 1)} \right]^T \cdot \frac{\partial V}{\partial \mathbf{x}(t_f)} \;\middle|\; \delta \mathbf{x}(0) \right\rangle \\
&= \left\langle \frac{\partial V}{\partial \mathbf{x}(0)} \;\middle|\; \delta \mathbf{x}(0) \right\rangle
\end{aligned}
$$

$$
\delta J_0 = \left\langle \frac{\partial V}{\partial \mathbf{x}(t_f)} \;\middle|\; \mathcal{TLM} \cdot \delta \mathbf{x}(0) \right\rangle = \left\langle \mathcal{ADM} \cdot \frac{\partial V}{\partial \mathbf{x}(t_f)} \;\middle|\; \delta \mathbf{x}(0) \right\rangle
$$

Compare $\mathcal{ADM}$ with expression for Lagrange multipliers.

3-box model of the THC

Inspired by work with Laure Zanna & Eli Tziperman (2010)

3-box model of the THC: overview

```
DO t = 1, nTimeSteps
```

- calc. density

$$\rho = -\alpha T + \beta S$$

- calc. thermohaline transport

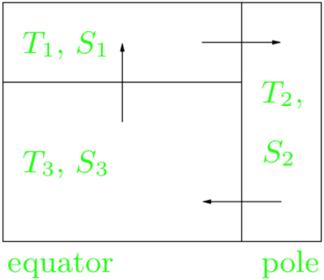
$$U = U(\rho(T, S))$$

- calc. tracer advection

$$\frac{d}{dt} Tr = f(Tr, U)$$

- calc. timestepping, update tracer fields $Tr = \{T, S\}$

```
END DO
```



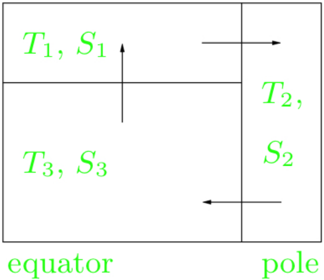
3-box model of the THC: overview

$$\rho_i = -\alpha T_i + \beta S_i$$

$$U = u_0 \Big\{ \rho_2 - [\, H\rho_1 + (1 - H)\rho_3 \,] \Big\}$$

$$\frac{dT_3}{dt} = U(T_2 - T_3)/V_3$$

$$\frac{dS_3}{dt} = U(S_2 - S_3)/V_3$$



$$\frac{dT_3}{dt} = U(T_3 - T_2), \quad \text{for } U \geq 0$$

$$\mathtt{diffT3} = \mathtt{u} * (\mathtt{T3} - \mathtt{T2})$$

- total derivative:

$$\delta\mathtt{diffT3} = \frac{\partial\mathtt{diffT3}}{\partial\mathtt{U}}\delta\mathtt{U} + \frac{\partial\mathtt{diffT3}}{\partial\mathtt{T}_2}\delta\mathtt{T}_2 + \frac{\partial\mathtt{diffT3}}{\partial\mathtt{T}_3}\delta\mathtt{T}_3$$

- in matrix form:

$$\begin{pmatrix} \delta\mathtt{diffT3} \\ \delta\mathtt{T}_3 \\ \delta\mathtt{T}_2 \\ \delta\mathtt{U} \end{pmatrix}^{\lambda} = \begin{pmatrix} 0 & -\mathtt{U} & \mathtt{U} & \mathtt{T3} - \mathtt{T1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \delta\mathtt{diffT3} \\ \delta\mathtt{T}_3 \\ \delta\mathtt{T}_2 \\ \delta\mathtt{U} \end{pmatrix}^{\lambda-1}$$

- Transposed relationship yields:

$$\begin{pmatrix} \delta^*\mathtt{diffT3} \\ \delta^*\mathtt{T}_3 \\ \delta^*\mathtt{T}_2 \\ \delta^*\mathtt{U} \end{pmatrix}^{\lambda-1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ -\mathtt{U} & 1 & 0 & 0 \\ \mathtt{U} & 0 & 1 & 0 \\ \mathtt{T3}-\mathtt{T1} & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \delta^*\mathtt{diffT3} \\ \delta^*\mathtt{T}_3 \\ \delta^*\mathtt{T}_2 \\ \delta^*\mathtt{U} \end{pmatrix}^{\lambda}$$

- and thus adjoint code:

```
adT3      =  adT3  -          u*addiffT3
adT2      =  adT2  +          u*addiffT3
adU       =  adu   +  (T3-T2)*addiffT3
addiffT3  =  0
```

Note: state $\mathtt{T2}, \mathtt{T3}, \mathtt{U}$ are required to evaluate derivative at each time step, in reverse order!
$\longrightarrow$ *TANGENT* linearity

Reverse order integration (i)

```
DO istep = 1, nTimeSteps
```

- `call density`(ρ)
- `call transport`(U)
- `call timestep`(T, S)
- `call update`(T, S)

```
END DO
```

```
DO istep = nTimeSteps, 1, -1
```

C recompute required variables

- `DO iaux = 1, istep`
  - `call density`(ρ)
  - `call transport`(U)
  - `call timestep`(T, S)
  - `call update`(T, S)

  `END DO`

C perform adjoint timestep

- `call adupdate`(T, S)
- `call adtimestep`(T, S)
- `call adtransport`(U)
- `call addensity`(ρ)

END DO

Reverse order integration (ii)

```
DO iOuter = 1, nOuter
```

- *CADJ STORE* $T, S \rightarrow$ *disk*
- `DO iInner = 1, nInner`
  - `call density(`ρ`)`
  - `call transport(`U`)`
  - `call timestep(`Tr`)`
  - `call update(`Tr`)`

  `END DO`

`END DO`

```
DO iOuter = nOuter, 1, -1
```

- *CADJ RESTORE* $T, S \leftarrow$ *disk*
- `DO iInner = 1, nInner`
  - `call density(`ρ`)`
  - `call transport(`U`)`
  - *CADJ STORE* T, S, U
  - `call timestep(`Tr`)`
  - `call update(`Tr`)`

  `END DO`
- `DO iInner = nInner, 1, -1`
  - `call adupdate(`$\mathrm{ad}Tr$`)`
  - `call adtimestep(`$\mathrm{ad}Tr$`)`
  - *CADJ RESTORE* T, S, U
  - `call adtransport(`$\mathrm{ad}U$`)`
  - `call addensity(`$\mathrm{ad}\rho$`)`

  `END DO`

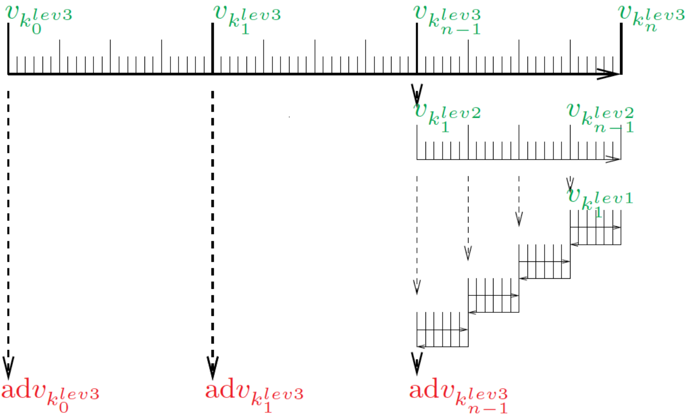
`END DO`

Reverse order integration (iii)

- *Adjoint evaluated in reverse*
  - → model state at every time step required in reverse
  - → all state stored or recomputed
- *Solution:* **Checkpointing**

  e.g. Griewank (1992), Retrepo et al. (1998)

  storing vs. recomputation



Reverse order integration (iv)

- e.g. 3-level checkpointing:

$$n_{TimeSteps} = n_1 \cdot n_2 \cdot n_3$$

$\rightarrow$ **Storing:** reduced from $n_1 \cdot n_2 \cdot n_3$ to
  - disk: $n_2 + n_3$,
  - memory: n_1

$\rightarrow$ **CPU:** $3 \cdot \mathrm{forward} + 1 \cdot \mathrm{adjoint} \approx 5.5 \cdot \mathrm{forward}$

- Closely related to **adjoint dump & restart** problem. Available queue sizes at HPC Centres may be limited

- Insertion of store directive requires detailed knowledge of code *and* AD tool behaviour
  $\longrightarrow$ not easy ("semi-automatic" differentiation only)

Ensure correctness of TLM or ADM derived gradient

Procedures to check that AD-derived gradient G_i^{ad} is correct:
consider perturbation of i-th control vector element u_i and $\Delta\vec{u}_i = \delta_{ij}$

finite difference vs. adjoint	tangent linear vs. adjoint
$G_i^{fd} = \frac{\mathcal{J}(u_i+\epsilon)-\mathcal{J}(u_i-\epsilon)}{2\epsilon}$	$G_i^{tl} = \vec{\nabla}_u\mathcal{J}\cdot\Delta\vec{u}_i = \left(\vec{\nabla}_u\mathcal{J}\right)_i$
$R_i^{fd} = 1 - \frac{G_i^{fd}}{G_i^{ad}}$	$R_i^{tl} = 1 - \frac{G_i^{tl}}{G_i^{ad}}$

$\rightarrow$ can test 'correctness' of ADM and TLM gradients G_i^{ad} G_i^{ad}

$\rightarrow$ can test 'time horizon' of linearity assumption

Other approaches: e.g., Taylor remainder test (Patrick Farrell)

Input/Output — active file handling

I/O of active variables should be accounted for in derivative

READ assigning a value to a variable

WRITE referencing a variable

code	hypothetical code	adjoint hypothetical code	adjoint code
`OPEN(8)`		`ADXD = 0.`	`OPEN(9)`
⋮	⋮	⋮	⋮
`WRITE(8) X`	`XD = X`	`ADXD = ADXD + ADZ` `ADZ = 0.`	`WRITE(9) ADZ` `ADZ = 0.`
⋮	⋮	⋮	⋮

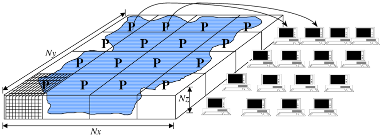
`READ(8) Z`	`Z = XD`	`ADX = ADX + ADXD`
		`ADXD = 0.`
			`READ(9) ADXD`
			`ADX = ADX + ADXD`
			`ADXD = 0.`

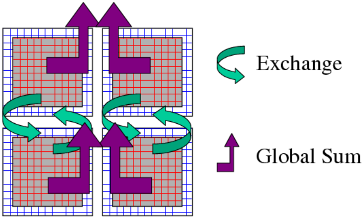
⋮	⋮	⋮	⋮
`CLOSE(8)`			`CLOSE(9)`

from *Giering & Kaminski (1998)*

Scalability

- domain decomposition (tiles) & overlaps (halos)
- split into extensive on-processor and global phase





Global communication/arithmetic op.'s supported by MITgcm's intermediate layer (WRAPPER) which need hand-written adjoint forms

	operation/primitive	forward		reverse
•	communication (MPI,...):	send	$\longleftrightarrow$	receive
•	arithmetic (global sum,...):	gather	$\longleftrightarrow$	scatter
•	active parallel I/O:	read	$\longleftrightarrow$	write

Why Algorithmic/Automatic Differentiation (AD)?

- Inaccuracy of finite-differences has negative impact on convergence of your algorithm.
  You need exact derivatives.
- The dimension ($n \sim O(10^8)$) of your problem is too lage for finite-difference or directional derivatives (tangent linear model) to be feasible approaches.
  You need efficient/cheap gradients.
- Your numerical code changes over time due to improvements, restructuring, new insights resulting from ongoing research and development. The corresponding derivative codes need to be updated too. Many man hours may be involved (and error-prone).
  You need a derivative code "compiler".
- Changing the control space (i.e. type of independent variables) may change structure of the derivative code.
  You need automated way to re-generate derivative code.