Tutorial 11:
Observation Impact &
Observation Sensitivity

Impact/Sensitivity Functions

We will use as an example, the time-averaged transport along a line of constant latitude, ϕ:

$$I = \frac{1}{(t_2 - t_1)} \int_{t_1}^{t_2} \int_{\lambda_1}^{\lambda_2} \int_{z_1}^{z_2} v(\lambda, \phi, z) dz d\lambda dt$$

$$\equiv \frac{1}{(k_2 - k_1)} \sum_{k=k_1}^{k_2} \mathbf{h}_k^{\mathrm{T}} \mathbf{x}_k$$

where $[t_1, t_2] \equiv [k_1 \Delta t, k_2 \Delta t]$ **and** $\mathbf{x}_k \equiv \mathbf{x}(k\Delta t)$

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{T}_k \\ \mathbf{S}_k \\ \varsigma_k \\ \mathbf{u}_k \\ \mathbf{v}_k \end{bmatrix} \qquad \mathbf{h}_k = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \Delta z \Delta \lambda \end{bmatrix} \qquad \frac{\partial I}{\partial \mathbf{x}_j} = \frac{1}{(k_2 - k_1)} \mathbf{h}_j$$

Impact/Sensitivity Functions

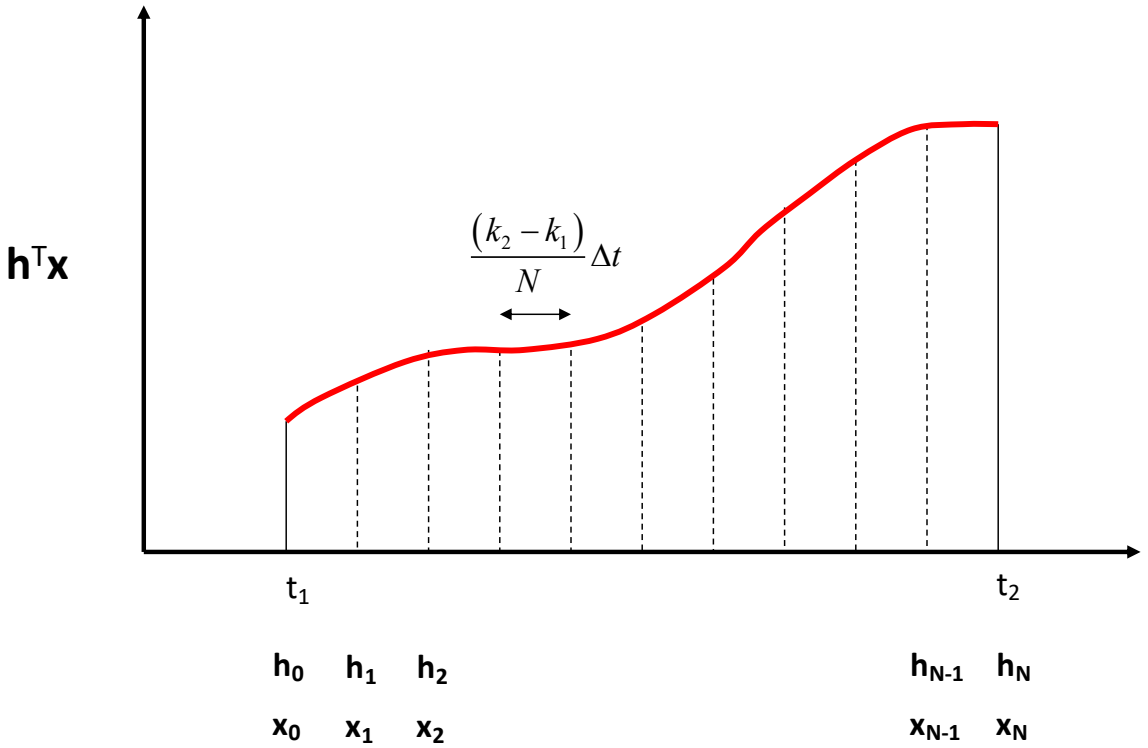
Recall that observation impact and observation sensitivity calculations involve *forcing* ADROMS with ∂*I*/∂x.

$$I = \frac{1}{\left(k_2 - k_1\right)} \sum_{k=k_1}^{k_2} \mathbf{h}_k^{\mathrm{T}} \mathbf{x}_k$$

$$\frac{\partial I}{\partial \mathbf{x}_j} = \frac{1}{\left(k_2 - k_1\right)} \mathbf{h}_j$$

The Trapezoidal Rule

Saving x every timestep is costly, so instead use an approx.

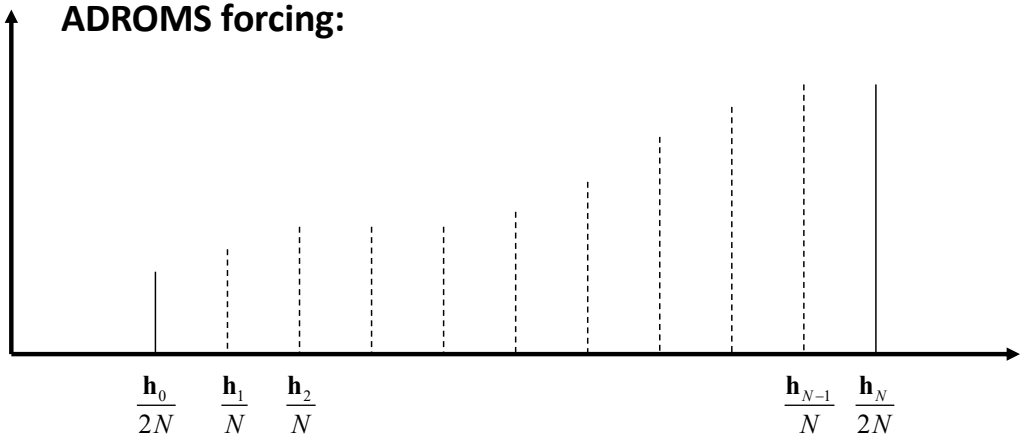


The Trapezoidal Rule

$$I = \frac{1}{N}\left(\frac{1}{2}\left(\mathbf{h}_0^{\mathrm{T}}\mathbf{x}_0 + \mathbf{h}_N^{\mathrm{T}}\mathbf{x}_N\right) + \sum_{i=1}^{N-1}\mathbf{h}_i^{\mathrm{T}}\mathbf{x}_i\right)$$

$$\frac{\partial I}{\partial \mathbf{x}_0} = \frac{1}{2N}\mathbf{h}_0; \quad \frac{\partial I}{\partial \mathbf{x}_N} = \frac{1}{2N}\mathbf{h}_N; \quad \frac{\partial I}{\partial \mathbf{x}_i} = \frac{1}{N}\mathbf{h}_i;$$

ADROMS forcing:



define AD_IMPULSE

Some Common General Cases

(i) Linear *I*:

$$I = \sum_{k=k_1}^{k_2} \mathbf{h}_k^{\mathrm{T}} \mathbf{x}_k ; \quad \partial \mathrm{I} / \partial \mathbf{x}_k = \mathbf{h}_k$$

(i) Quadratic *I*:

$$I = \sum_{k=k_1}^{k_2} \left(\mathbf{x}_k \pm \mathbf{g} \right)^{\mathrm{T}} \mathbf{L} \left(\mathbf{x}_k \pm \mathbf{g} \right); \quad \partial \mathrm{I} / \partial \mathbf{x}_k = \left(\mathbf{L} + \mathbf{L}^{\mathrm{T}} \right) \left(\mathbf{x}_k \pm \mathbf{g} \right)$$

where $\mathbf{g}$ is an arbitrary vector that is not a function of x.

Practical Matters: How to do it yourself

First, write a matlab script to compute the functional *I* of interest.

Important considerations:

- Abandon fancy matlab programming – keep it simple!
- Avoid intrinsic matlab functions, structures and cell arrays (at least until you know what you are doing!)
- Use “for-loops” for transparency

Writing Adjoint Operators

Recall that what we need to run the adjoint model is $\partial I / \partial \mathbf{x}$

So we need is a method for differentiating a matlab script.

A useful result is that if **y=Ax**, then d**y**/d**x**=**A**^{T}

A fool-proof recipe for differentiating code (Giering and Kaminski, 1998):

Matlab code to compute y=Ax

```
x(1:N)=input;
y(1:M)=zeros(M,1);
for i=1:M
  for j=1:N
    y(i)=y(i)+a(i,j)*x(j);
  end
end
y(1:M)=>output
```

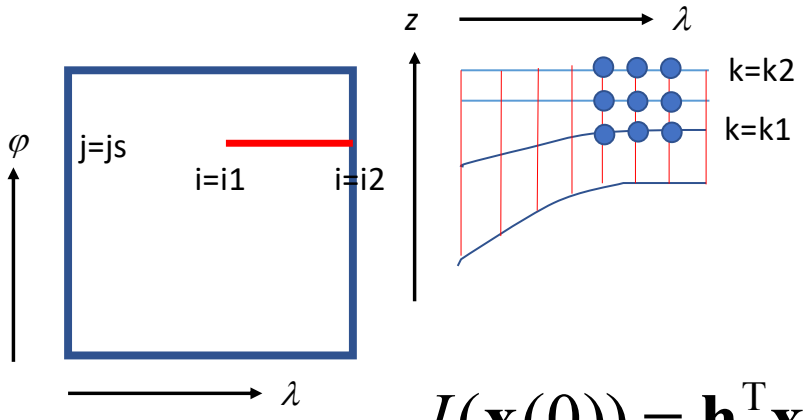
Matlab code to compute x =A^{T}y

```
ad_y(1:M)=input;
ad_x=zeros(N,1);
for i=1:M
  for j=1:N
    ad_x(j)=ad_x(j)+a(i,j)*ad_y(i);
  end
End
ad_y(1:M)=zeros(M,1);
ad_x(1:N)=>output
```

This represents the derivative of "$y_i=y_i+a_{ij}x_j$" wrt to x_j.
Step 1: $d/dx_j(y_i+a_{ij}x_j)=a_{ij}$.
Step 2: Multiply this derivative by the adjoint of the variable on the rhs, ad_y_j in this case, a_{ij}*ad_y_j.
Step 3: Keep a running sum in case x_j is used elsewhere later, $ad\_x_j=ad\_x_j+a_{ij}$*ad_y_j

An Illustrative Example: Transport

```
rec=1;
v=nc_read('history.nc','v',rec);
js=??;
trans=0;
for k=k1:k2
   for i=i1:i2
      fac=dlamba(i,js)*dz(i,js,k);
      trans=trans+fac*v(i,js,k);
    end
end
trans=>output
```



$$I(\mathbf{x}(0)) = \mathbf{h}^{\mathrm{T}}\mathbf{x}(0)$$

The next step is to create appropriate forcing fields for the adjoint model. This means we need the derivative of our matlab script.

An Illustrative Example: Transport

Matlab code to compute transport

```
rec=1;
v=nc_read('history.nc','v',rec);
js=??;
trans=0;
for k=k1:k2
   for i=i1:i2
     fac=dlamba(i,js)*dz(i,js,k);
     trans=trans+fac*v(i,js,k);
    end
end
trans=>output
```

Matlab code to compute h for ADSname.nc (Work backwards when deriving this)

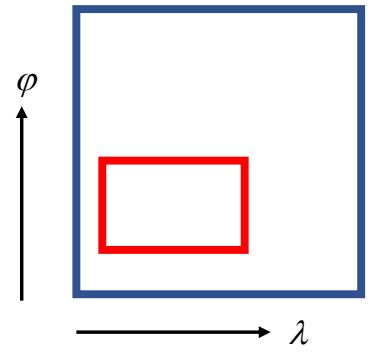
```
rec=1;
v=nc_read('history.nc','v',rec);
js=??;
ad_v=zeros(size(v));
ad_trans=1;
for k=k1:k2
   for i=i1:i2
     fac=dlamba(i,js)*dz(i,js,k);
     ad_v(i,is,k)=ad_v(i,js,k)+fac*ad_trans;
    end
end
nc_write('ads.nc','v',ad_v,rec);
```

$$I(\mathbf{x}(0)) = \mathbf{h}^{\mathrm{T}}\mathbf{x}(0)$$

An Illustrative Example: Eddy Kinetic Energy

```
rec=1;
rho=1025;
u=nc_read('history.nc','u',rec);
v=nc_read('history.nc','v',rec);
uc=nc_read('climatology.nc','u',rec);
vc=nc_read('climatology.nc','v',rec);
eke=0;
for k=k1:k2
  for j=j1:j2
    for i=i1:i2
      fac=dlamba(i,j)*dphi(i,j)*dz(i,j,k);
      du=fac*(u(i,j,k)-uc(i,j,k));
      dv=fac*(v(i,j,k)-vc(i,j,k));
      eke=eke+du*du+dv*dv;
    end
  end
end
eke=0.5*rho*eke;
eke=>output
```

$$I\left(\mathbf{x}(0)\right) = \mathbf{x}(0)^{\mathrm{T}}\mathbf{E}\mathbf{x}(0)$$



Since the functional is non-linear in x, we need to first linearize. The idea here is that:

$$\partial\mathbf{x}^{\mathrm{T}}\mathbf{E}\mathbf{x}/\partial\mathbf{x} = \mathbf{E}\mathbf{x} + \mathbf{E}^{\mathrm{T}}\mathbf{x} = 2\mathbf{E}^{\mathrm{T}}\mathbf{x}$$

An Illustrative Example: Eddy Kinetic Energy

Code to compute EKE

```
rec=1;
rho=1025;
u=nc_read('history.nc','u',rec);
v=nc_read('history.nc','v',rec);
uc=nc_read('climatology.nc','u',rec);
vc=nc_read('climatology.nc','v',rec);
eke=0;
for k=k1:k2
  for j=j1:j2
    for i=i1:i2
      fac=dlamba(i,j)*dphi(i,j)*dz(i,j,k);
      du=fac*(u(i,j,k)-uc(i,j,k));
      dv=fac*(v(i,j,k)-vc(i,j,k));
      eke=eke+du*du+dv*dv;
    end
  end
end
eke=0.5*rho*eke;
eke=>output
```

Code to compute tangent linear EKE

```
rec=1;
rho=1025;
u=nc_read('history.nc','u',rec);
v=nc_read('history.nc','v',rec);
uc=nc_read('climatology.nc','u',rec);
vc=nc_read('climatology.nc','v',rec);
tl_eke=0;
for k=k1:k2
  for j=j1:j2
    for i=i1:i2
      fac=dlamba(i,j)*dphi(i,j)*dz(i,j,k);
      du=fac*(u(i,j,k)-uc(i,j,k));
      dv=fac*(v(i,j,k)-vc(i,j,k));
      tl_eke=tl_eke+2*(tl_u(i,j,k)*du+2*tl_v(i,j,k)*dv);
    end
  end
end
tl_eke=0.5*rho*tl_eke;
tl_eke=>output
```

You are never going to run this - it is an intermediate step to deriving the adjoint code.

Code to compute tangent linear EKE

```
rec=1;
rho=1025;
u=nc_read('history.nc','u',rec);
v=nc_read('history.nc','v',rec);
uc=nc_read('climatology.nc','u',rec);
vc=nc_read('climatology.nc','v',rec);
tl_eke=0;
for k=k1:k2
  for j=j1:j2
    for i=i1:i2
      fac=dlamba(i,j)*dphi(i,j)*dz(i,j,k);
      du=fac*(u(i,j,k)-uc(i,j,k));
      dv=fac*(v(i,j,k)-vc(i,j,k));
      tl_eke=tl_eke+2*fac*tl_u(i,j,k)*du+2*fac*tl_v(i,j,k)*dv;
    end
  end
end
tl_eke=0.5*rho*tl_eke;
tl_eke=>output
```

Code to compute the input for the adjoint model (WORK BACKWARDS)

```
rec=1;
rho=1025;
u=nc_read('history.nc','u',rec);
v=nc_read('history.nc','v',rec);
uc=nc_read('climatology.nc','u',rec);
vc=nc_read('climatology.nc','v',rec);
ad_u=zeros(size(u));
ad_v=zeros(size(v));
ad_eke=1;
ad_eke=0.5*rho*ad_eke;
for k=k1:k2
  for j=j1:j2
    for i=i1:i2
      fac=dlamba(i,j)*dphi(i,j)*dz(i,j,k);
      du=fac*(u(i,j,k)-uc(i,j,k));
      dv=fac*(v(i,j,k)-vc(i,j,k));
      ad_u(i,j,k)=ad_u(i,j,k)+2*fac*du*ad_eke;
      ad_v(i,j,k)=ad_v(i,j,k)+2*fac*dv*ad_eke;
    end
  end
end
nc_write('ads.nc','u',ad_u,rec);
nc_write('ads.nc','v',ad_v,rec);
```

Code to compute EKE

```
rec=1;
rho=1025;
u=nc_read('history.nc','u',rec);
v=nc_read('history.nc','v',rec);
uc=nc_read('climatology.nc','u',rec);
vc=nc_read('climatology.nc','v',rec);
eke=0;
for k=k1:k2
  for j=j1:j2
    for i=i1:i2
      fac=dlamba(i,j)*dphi(i,j)*dz(i,j,k);
      du=fac*(u(i,j,k)-uc(i,j,k));
      dv=fac*(v(i,j,k)-vc(i,j,k));
      eke=eke+du*du+dv*dv;
    end
  end
end
eke=0.5*rho*eke;
eke=>output
```

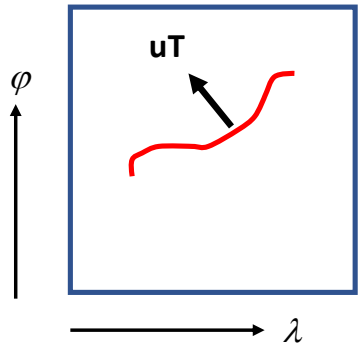
Code to compute the input for the adjoint model (WORK BACKWARDS)

```
rec=1;
rho=1025;
u=nc_read('history.nc','u',rec);
v=nc_read('history.nc','v',rec);
uc=nc_read('climatology.nc','u',rec);
vc=nc_read('climatology.nc','v',rec);
ad_u=zeros(size(u));
ad_v=zeros(size(v));
ad_eke=1;
ad_eke=0.5*rho*ad_eke;
for k=k1:k2
  for j=j1:j2
    for i=i1:i2
      fac=dlamba(i,j)*dphi(i,j)*dz(i,j,k);
      du=fac*(u(i,j,k)-uc(i,j,k));
      dv=fac*(v(i,j,k)-vc(i,j,k));
      ad_du=ad_du+2*du*ad_eke;
      ad_dv=ad_dv+2*dv*ad_eke;
      ad_v(i,j,k)=ad_v(i,j,k)+fac*ad_dv;
      ad_dv=0;
      ad_u(i,j,k)=ad_u(i,j,k)+fac*ad_du;
      ad_du=0;
    end
  end
end
nc_write('ads.nc','u',ad_u,rec);
nc_write('ads.nc','v',ad_v,rec);
```

A more complex example

Code to compute heat flux normal to an arbitrary vertical section

```
Ginp=roms_getgrid('history.nc');
temp=nc_read('history.nc','temp');
u=nc_read('history.nc','u');
v=nc_read('history.nc','v');
[A,B]=roms_genslice(f'history.nc','temp',lonTrk,latTrk);
np=size(lonTrk,1);
en=B.en;
T=interpolator(Ginp,temp,lonTrk,latTrk);
Us=interpolator(Ginp,u,lonTrk,latTrk);
Vs=interpolator(Ginp,v,lonTrk,latTrk);
Vn=real(conj(en)*complex(Us,Vs));
hf=0;
area=0;
for k=k1:k2
   for i=1:np
      area=area+ds(i)*dz(i,k);
      hf=hf+T(i)*Vn(i)*ds(i)*dz(i,k);
   end
end
hf=rho*Cp*hf/area;
hf=>output
```



roms_genslice, interpolator, and ad_interpolator are available in the ROMS matlab repository.

Code to compute heat flux normal to an arbitrary vertical section

```
Ginp=roms_getgrid('history.nc');
temp=nc_read('history.nc','temp');
u=nc_read('history.nc','u');
v=nc_read('history.nc','v');
[A,B]=roms_genslice(f'history.nc','temp',lonTrk,latTrk);
np=size(lonTrk,1);
en=B.en;
T=interpolator(Ginp,temp,lonTrk,latTrk);
Us=interpolator(Ginp,u,lonTrk,latTrk);
Vs=interpolator(Ginp,v,lonTrk,latTrk);
Vn=real(conj(en)*complex(Us,Vs));
hf=0;
area=0;
for k=k1:k2
   for i=1:np
      area=area+ds(i)*dz(i,k);
      hf=hf+T(i)*Vn(i)*ds(i)*dz(i,k);
   end
end
hf=rho*Cp*hf/area;
hf=>output
```

Code to compute the input for the adjoint model

```
SAME PREAMBLE AS LEFT
ad_temp=zeros(size(temp));
ad_u=zeros(size(u));
ad_v=zeros(size(v));
ad_T=zeros(size(T));
ad_Vn=zeros(size(Vn));
ad_hf=rho*Cp/area;
for k=k1:k2
   for i=1:np
      ad_T(i)=ad_T(i)+ Vn(i)*ds(i)*dz(i,k)*ad_hf;
      ad_Vn(i)=ad_Vn(i)+T(i)*ds(i)*dz(i,k)*ad_hf;
   end
end
ad_Us=real(conj(en))*ad_Vn;
ad_Vs=imag(conj(en))*ad_Vn;
ad_u=ad_interpolator(Ginp,u,lonTrk,latTrk,ad_Us);
ad_v=ad_interpolator(Ginp,v,lonTrk,latTrk,ad_Vs);
ad_temp=ad_interpolator(Ginp,temp,lonTrk,latTrk,ad_T);
nc_write('ads.nc','u','ad_u);
nc_write('ads.nc','uv,'ad_v);
nc_write('ads.nc','temp','ad_temp);
```

cpp options and input parameters

```
#define RBL4DVAR_SENSITIVITY
#define OBS_IMPACT
#define OBS_IMPACT_SPLIT
#define AD_IMPULSE
```

ocean.in:

```
DstrSb=0;
DendSb=0;
KstrSb=1;
KendSb=# levels;

Lstate(isFsur) == T
Lstate(isUbar) == T
Lstate(isVbar) == T
Lstate(isUvel) == T
Lstate(isVvel) == T
Lstate(isTvar) == T T
```