Part 2:
Iterative solvers

M D Sacchi
University of Alberta

Course information

- Email: msacchi@ualberta.ca
- Web: https://sites.ualberta.ca/~msacchi/LAPIS/
- Approval: Assignment consisting of programming exercises due by April 30, 2019
- Delivery format: PDF by email

Forward and Adjoint Operators

- **Two special operations (or operators?)**

$$\textbf{Forward}: \mathbf{d} = \mathbf{Lm}$$

$$\textbf{Conjugate transpose or adjoint}: \tilde{\mathbf{m}} = \mathbf{L}'\mathbf{d}$$

- $\mathbf{L}$ is a linear operator (Forward modelling operator).

- Why are them special?

  - Iterative solvers for large inverse problems only need to know how to evaluate $\mathbf{L}[\,\cdot\,]$ and $\mathbf{L}'[\,\cdot\,]$

  - I usually interpret operators as matrices. In reality, operators are codes applied *on the flight*

  - *We will see these operators everywhere today*

Steepest descent method

- Compute the gradient and iterate downhill until convergence

- Let's see the special role of $\mathbf{L}$ and $\mathbf{L}'$ in steepest descent optimization (or in any iterative optimization algorithm that only requires $\mathbf{L}$ and $\mathbf{L}'$)

- Simplify problem by considering minimization of quadratic cost J

$$J = \|\mathbf{d} - \mathbf{L}\mathbf{m}\|_2^2$$

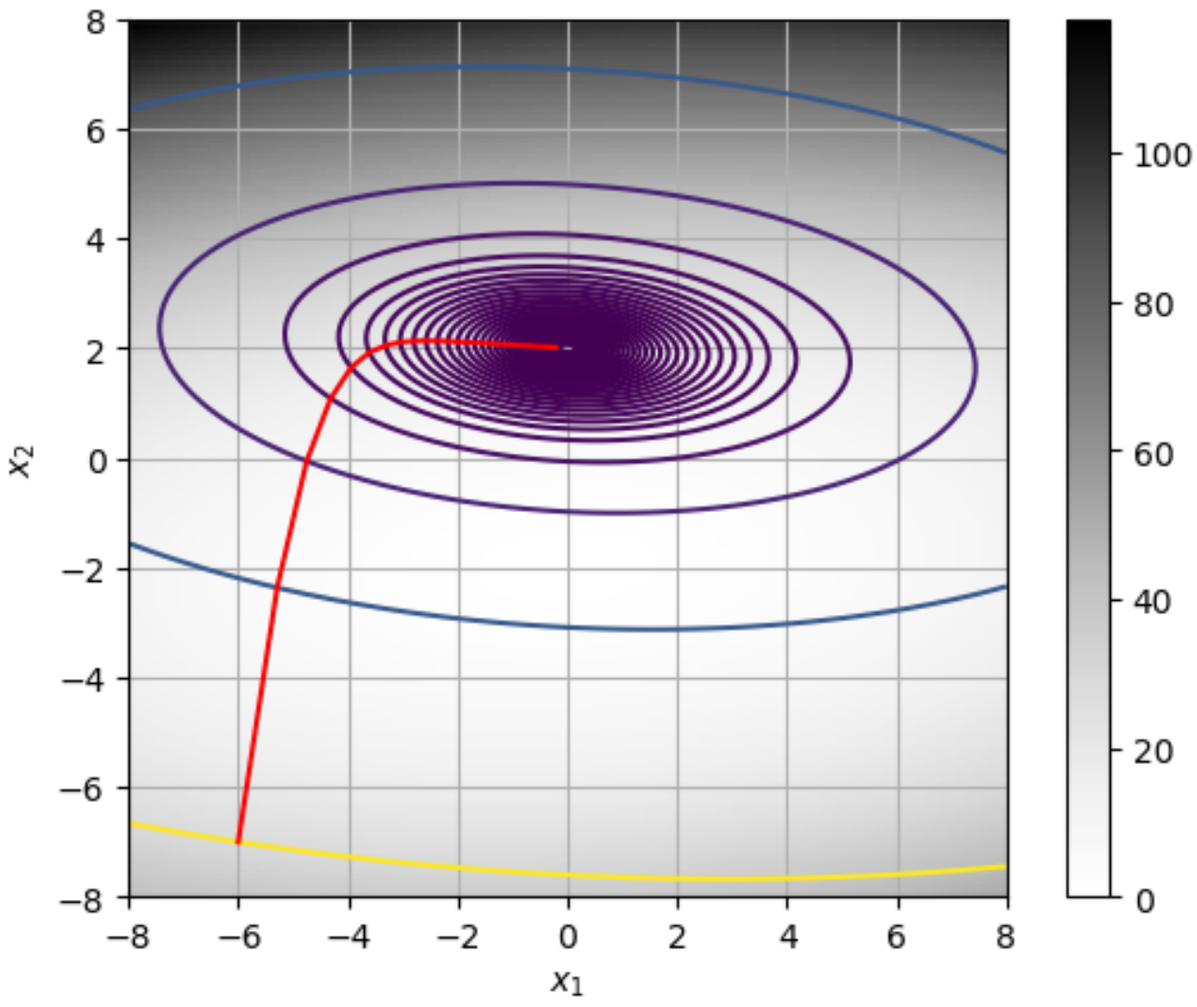
Steepest descent method

- Cost $J = \|\mathbf{d} - \mathbf{Lm}\|_2^2$
- Gradient $\mathbf{g} = \nabla J = 2\,\mathbf{L}'(\mathbf{Lm} - \mathbf{d})$
- Update $\mathbf{m}^{k+1} = \mathbf{m}^k - \alpha \mathbf{g}^k$
- Interesting, one can minimize J with a simple rule that does not involve inverting matrices and this is great because we can replace matrices by linear operators!!!!
- Aren't you excited ?

Steepest descent method

Demo_3_Lapis_2019_Steepest_Descent.ipynb

- Minimization of quadratic cost by SD



x_init = [-6.0,-7.9]
x_final=[-0.009,2.005]
x_true=[0.0,2.0]

K = 50 iterations at fixed step size

Steepest descent method

SD iterations:

Do until convergence

$$\mathbf{m}^{k+1} = \mathbf{m}^{k} - \alpha\, \mathbf{L}'(\mathbf{L}\mathbf{m}^{k} - \mathbf{d})$$

End Do

The above code can also be written as follows

Do until convergence

$$\mathbf{r}^{k} = (\mathbf{L}\mathbf{m}^{k} - \mathbf{d})$$

$$\mathbf{m}^{k+1} = \mathbf{m}^{k} - \alpha\, \mathbf{L}'(\mathbf{r}^{k})$$

End Do

Steepest descent method: Matrices are replaced by Linear Operators packed into Functions or Subroutines

Do until convergence

$$\mathbf{r}^k = (\mathbf{L}\mathbf{m}^k - \mathbf{d})$$

$$\mathbf{m}^{k+1} = \mathbf{m}^k - \alpha\, \mathbf{L}'(\mathbf{r}^k)$$

End Do

Do until convergence

$$\mathbf{r}^k = \textbf{Do\_It}\,[\mathbf{m}^k, flag = f\,] - \mathbf{d}$$

$$\mathbf{m}^{k+1} = \mathbf{m}^k - \alpha\, \textbf{Do\_It}[\mathbf{r}^k, flag = a]$$

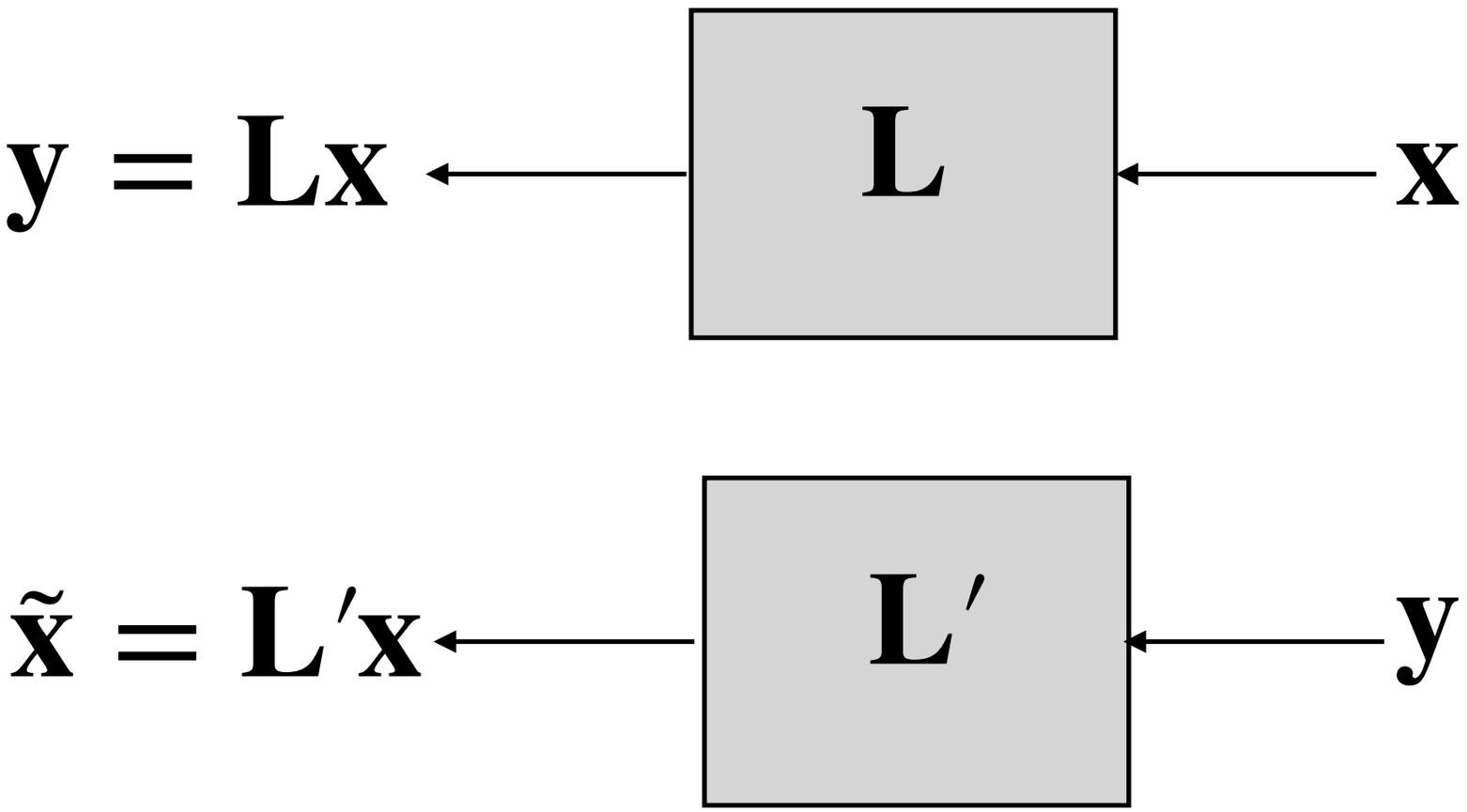
End Do

f: *Forward*

a: *Adjoint or transpose*

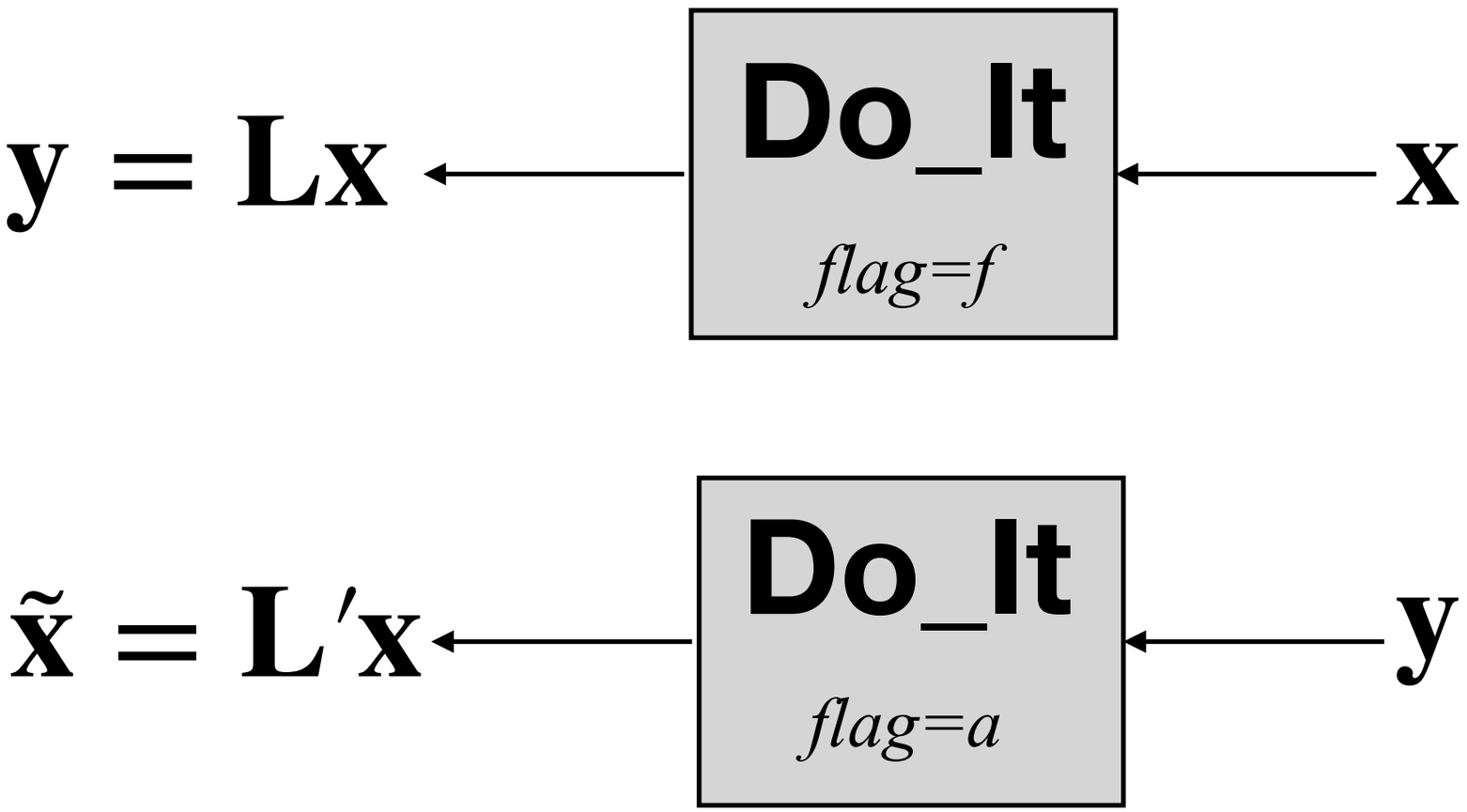
Operators *on the flight*

(al vuelo o pegarle de volea)



Operators *on the flight*

(al vuelo o pegarle de volea)



f: *Forward*

a: *Adjoint or transpose*

Why this is important?

- Migration and de-migration operators cannot be written as matrices. Least-squares migration requires the **Do_It** approach.

- Reconstruction problems often entail using operators that cannot be written via implicit matrices (FFTs, Curvelet Frames, etc). In this case, we also adopt the **Do_It** approach

- Multidimensional problems where $\mathbf{m}$ and $\mathbf{d}$ are not vectors can still be analyzed via linear algebra tools by providing simple rules that make linear operators behave like matrices

Dot-product test

- How do you guarantee that your codes for the forward and adjoint operators behave like $\mathbf{L}$ and $\mathbf{L}$' ?

- Let's $\mathbf{L}$ be the forward operator and let's call $\mathbf{B}$ the tentative adjoint or transpose operator

$$\mathbf{y}_1 = \mathbf{L}\,\mathbf{x}_1\,, \quad \mathbf{x}_2 = \mathbf{B}\,\mathbf{y}_2$$

- Form the two inner products

$$\mathbf{y}_2^T \mathbf{y}_1 = \mathbf{y}_2 \mathbf{L} \mathbf{x}_1$$

$$\mathbf{x}_1^T \mathbf{x}_2 = \mathbf{x}_1 \mathbf{B} \mathbf{y}_2$$

Dot-product test

The two inner products are equal

$$\mathbf{y}_2^T \mathbf{y}_1 = \mathbf{y}_2 \mathbf{L} \mathbf{x}_1 \qquad\qquad \mathbf{x}_1^T \mathbf{x}_2 = \mathbf{x}_1 \mathbf{B} \mathbf{y}_2$$

If $\mathbf{B} = \mathbf{L}'$

Therefore, one can write a code for $\mathbf{L}$ and code for $\mathbf{L}$', do the dot-product test and if the two product are equal then one can say that $\mathbf{L}$ and $\mathbf{L}$' packed by the function **Do_It** behave like a matrix and its transpose, respectively. Then you can safely use all you know about linear algebra to solve an inverse problem!

Dot-product test in practice

```
using LinearAlgebra, FFTW


# Dot product test example using operators rather than matrices
# Fourier DFT matrices and its Hermitian Transpose are replaced by
# on-the-flight FFTs
M = 512
x1 = randn(M)
y1 = fft(x1)
y2 = randn(M)
x2 = M*ifft(y2)
dot_x = x1'*x2
dot_y = y1'*y2
println(dot_x)
println(dot_y)
```

Demo_5_Lapis_2019.ipynb

Conjugate Gradients

- It is more efficient to use the Conjugate Gradient (CG) method than SD.

- I will not discuss CG but it basically amounts to also applying on the flight the operators $\mathbf{L}$ and $\mathbf{L'}$ in each iteration (step)

- The Conjugate Gradient algorithm is so popular that you can probably get one at a *Maxi Kiosko*

- *CG minimizes the general quadratic cost function:*

$$J = \|\mathbf{d} - \mathbf{Lm}\|_2^2$$

Concatenation of operators

- Forward

$$\mathbf{L} = \mathbf{ABC}$$

- Adjoint

$$\mathbf{L}' = \mathbf{C}'\mathbf{B}'\mathbf{A}'$$

- Dot product test must work for individual operators (codes) and then it will work for $\mathbf{L}$ and $\mathbf{L}'$

Preconditioning

- Using iterative methods (SD or CG) I prefer not to worry about matrices of weights

$$J = \|\mathbf{d} - \mathbf{Lm}\|_2^2 + \mu\|\mathbf{Wm}\|_2^2$$

- Therefore, I like to use the following change of variables

$$\mathbf{Wm} = \mathbf{u} \rightarrow \mathbf{Pu} = \mathbf{m}$$

$$J = \|\mathbf{d} - \mathbf{LPu}\|_2^2 + \mu\|\mathbf{u}\|_2^2$$

Preconditioning

- Minimize with iterative solver

$$J = \|\mathbf{d} - \mathbf{LPu}\|_2^2 + \mu\|\mathbf{u}\|_2^2$$

- With **Do_It** operators:

Forward : $\mathbf{LP}$

Adjoint : $\mathbf{P'L'}$

Preconditioning

- In this example, I use SD to minimize

$$J = \|\mathbf{d} - \mathbf{LPu}\|_2^2 + \mu\|\mathbf{u}\|_2^2$$

Do until convergence

$$\mathbf{r}^k = (\mathbf{LPu}^k - \mathbf{d})$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k - \alpha(\mathbf{P}'\mathbf{L}'(\mathbf{r}^k) + \mu\,\mathbf{u}^k)$$

End

$$\mathbf{m}_{sol} = \mathbf{Pu}$$

Preconditioning

- $\mathbf{W}$ and $\mathbf{P}$ should behave like the inverse of each other

$$\mathbf{Wm} = \mathbf{u} \rightarrow \mathbf{Pu} = \mathbf{m}$$

Then

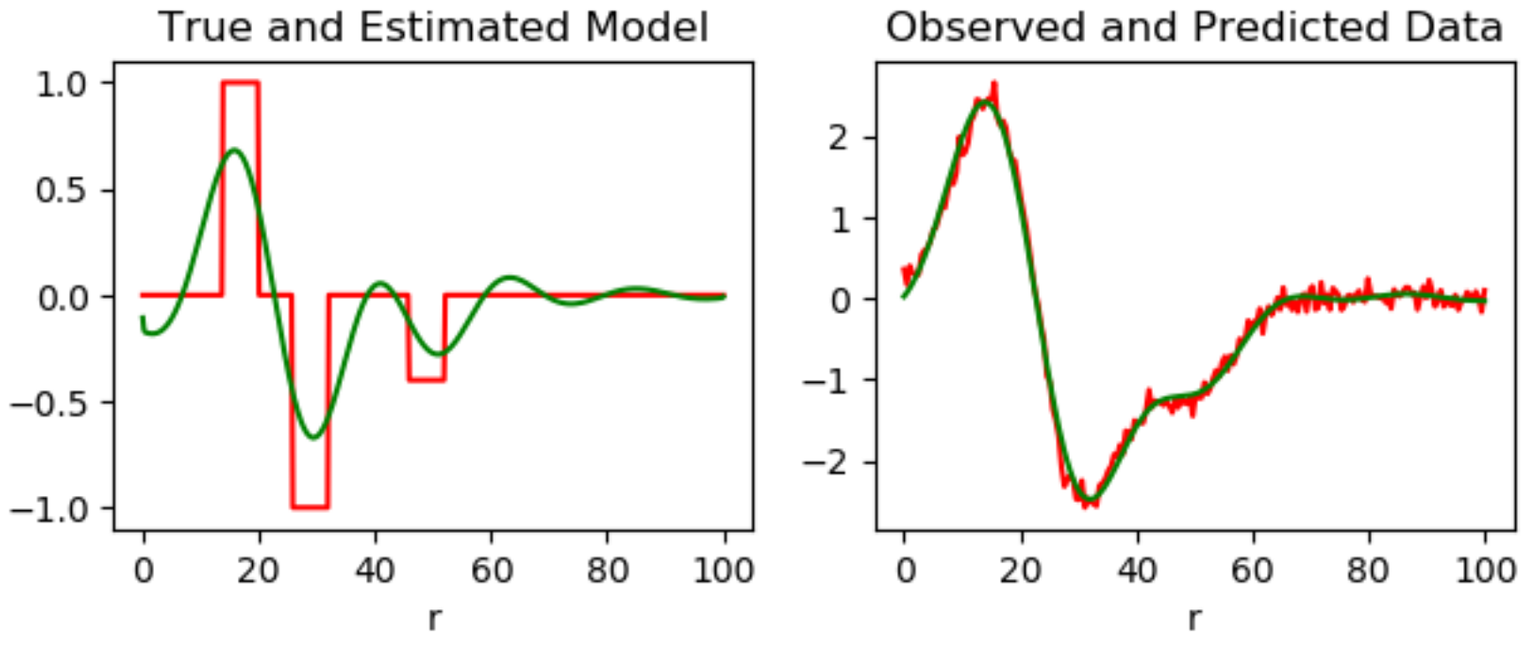
$\mathbf{W}$ is a high-pass operator (applies roughening)

$\mathbf{P}$ is low-pass operator (applies smoothing)

Preconditioning (SD)

Demo_4_Lapis_2019.ipynb

$$\mathbf{P} = \frac{1}{2}\begin{pmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$



ge-o-phys-i-cist, n.

A geophysicist is a person who passes as an exacting expert on the basis of being able to turn out, with prolific fortitude, infinite strings of incomprehensible formulae calculated with micrometric precision from vague assumptions, which are based on debatable figures taken from inconclusive experiments, carried out with instruments of problematic accuracy by persons of doubtful reliability and questionable morality for the avowed purpose of annoying and confounding a hopeless chimerical group of fanatics known as geologists who are themselves the lunatic fringe of the scientific community.

Author unknown