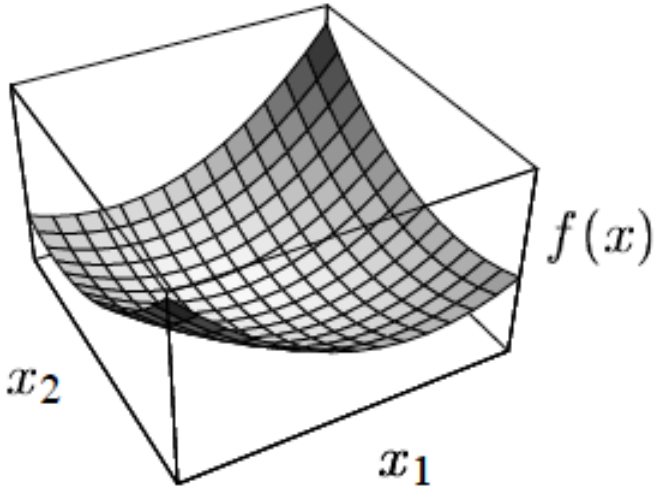
Conjugate gradients: ingredients

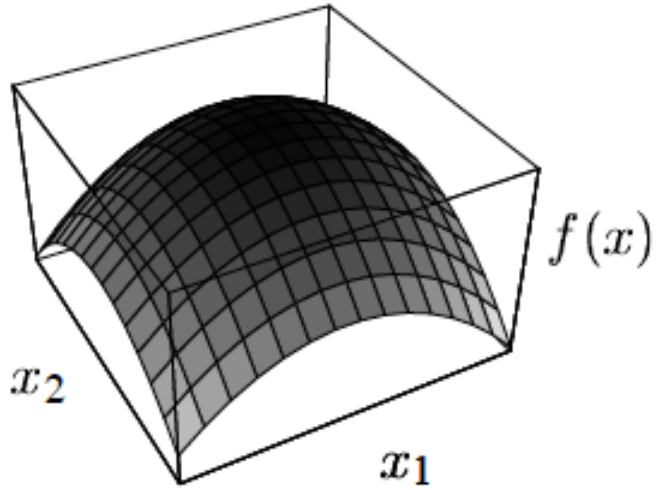
- Quadratic form
- Steepest descent
- Eigenvectors and Eigenvalues
- Conjugate directions

Quadratic forms of a matrix

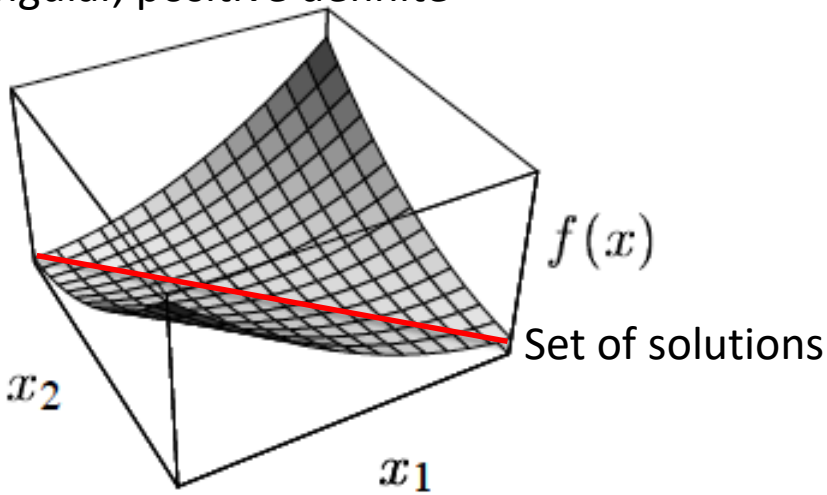
Positive definite



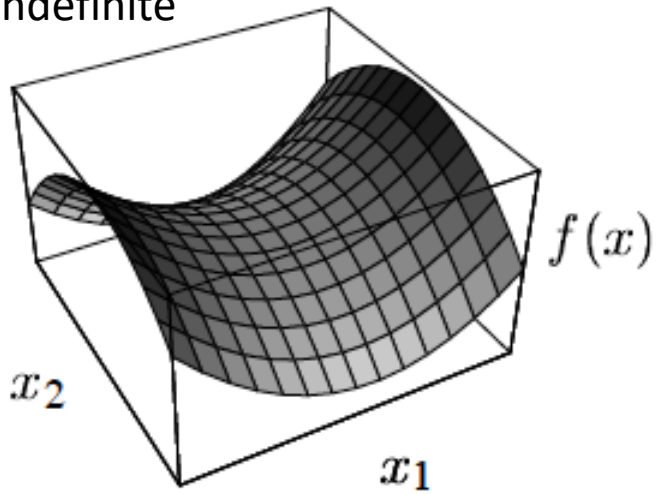
Negative definite



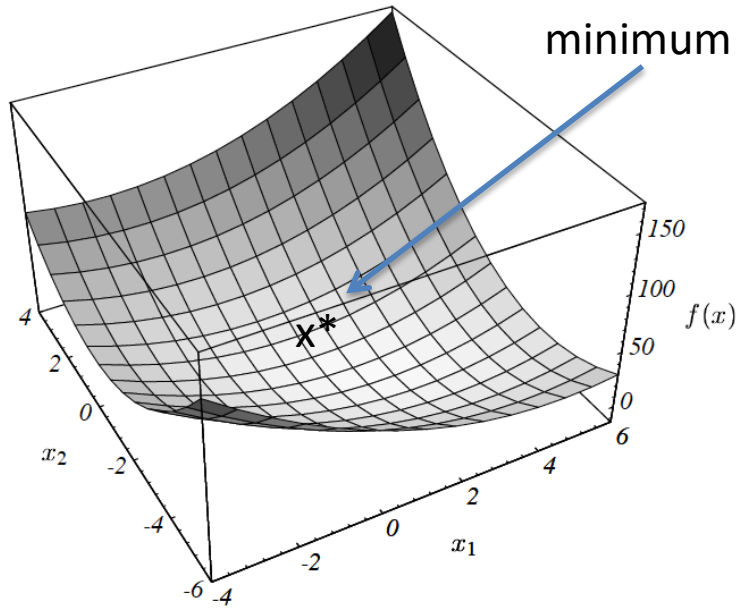
Singular, positive definite



Indefinite



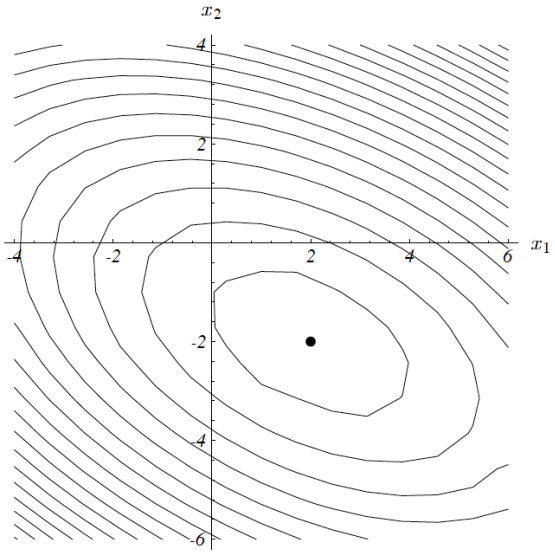
Quadratic forms

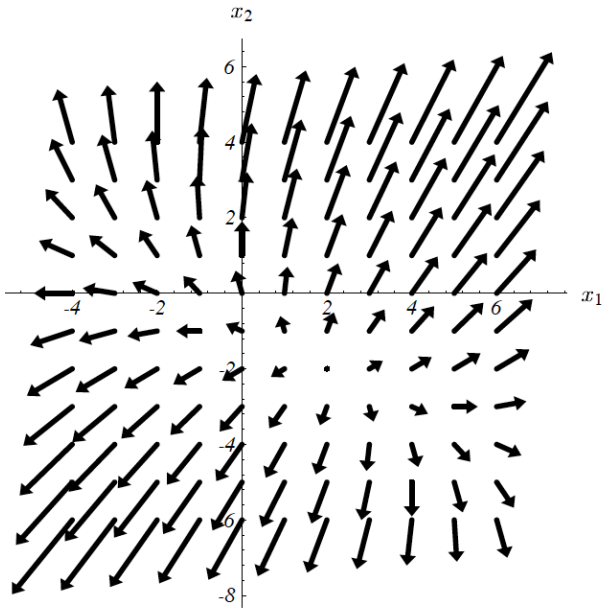


General quadratic equation

$$f(x) = \frac{1}{2} \mathrm{x}^{\mathrm{T}} \mathrm{A}\mathrm{x} - \mathrm{x}^{\mathrm{T}} \mathrm{b}$$

$f(x)$ is minimized by the solution to $\mathrm{A}\mathrm{x}=\mathrm{b}$, which is demonstrated on the next slide.





Gradient points in the direction of steepest increase of $f(x)$.

Conjugate gradient (cont'd)

Gradient

$$f'(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix}$$

With some more math we arrive at

$$f'(x) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathrm{A} + \frac{1}{2}(\mathrm{Ax})^{\mathrm{T}} - \mathrm{b}^{\mathrm{T}}$$

which, if A is symmetric, reduces to

$$f'(x) = \mathrm{Ax} - \mathrm{b}$$

Setting $f'(x)=0$, this is equation we need to solve (at the minimum, the first derivative is zero).

Thus, $\mathrm{Ax}=\mathrm{b}$ can be solved by finding x that minimizes $f(x)$.This holds, if A is symmetric and positive definite!

Steepest descent

Question: How to get to the minimum (fast, efficiently)?

This is a general question, which arises in many different contexts, such as solving PDEs and inversion.

Suggestion: slide down to the minimum from an arbitrary starting point x_0 following a number of steps, x_1, x_2,…., x_n until we are close enough to the solution x.

$$-f'(x_i) = b - Ax_i$$

This would be the direction to follow in which f(x) decreases fastest (direction of steepest descent).

$$e_i = x_i - x$$

This is the **error vector**, which indicates the distance from the true solution vector, x, at each step, $i=1,\ldots,n$.

$$r_i = b - Ax_i$$

$$r_i = -f'(x_i)$$

This is the **residual vector**, which indicates the distance from the true b vector at each step, $i=1,\ldots,n$. Think of it as the error vector transformed by A into the b-space. The residual vector is actually the direction of the steepest descent!

Steepest descent (cont'd)

These are then the different steps of the Steepest Descent method:

$$r_i = b - Ax_i$$

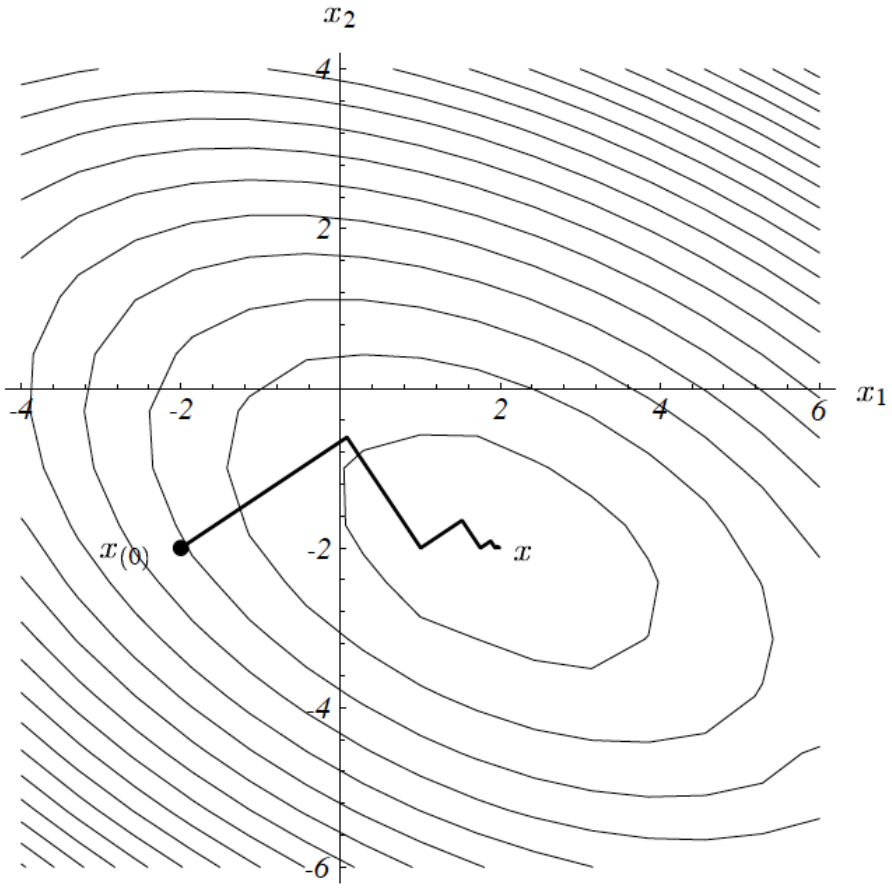
$$\alpha_i = \frac{r_i^T r_i}{r_i^T A r_i}$$

$$x_{i+1} = x_i + \alpha_i r_i$$

The method requires two matrix-vector products per iteration! One can be eliminated though:

$$r_{i+1} = r_i - \alpha_i A r_i$$

Remember i and i+1 are the iteration counts of the steepest descent solver (NOT spatial indices)



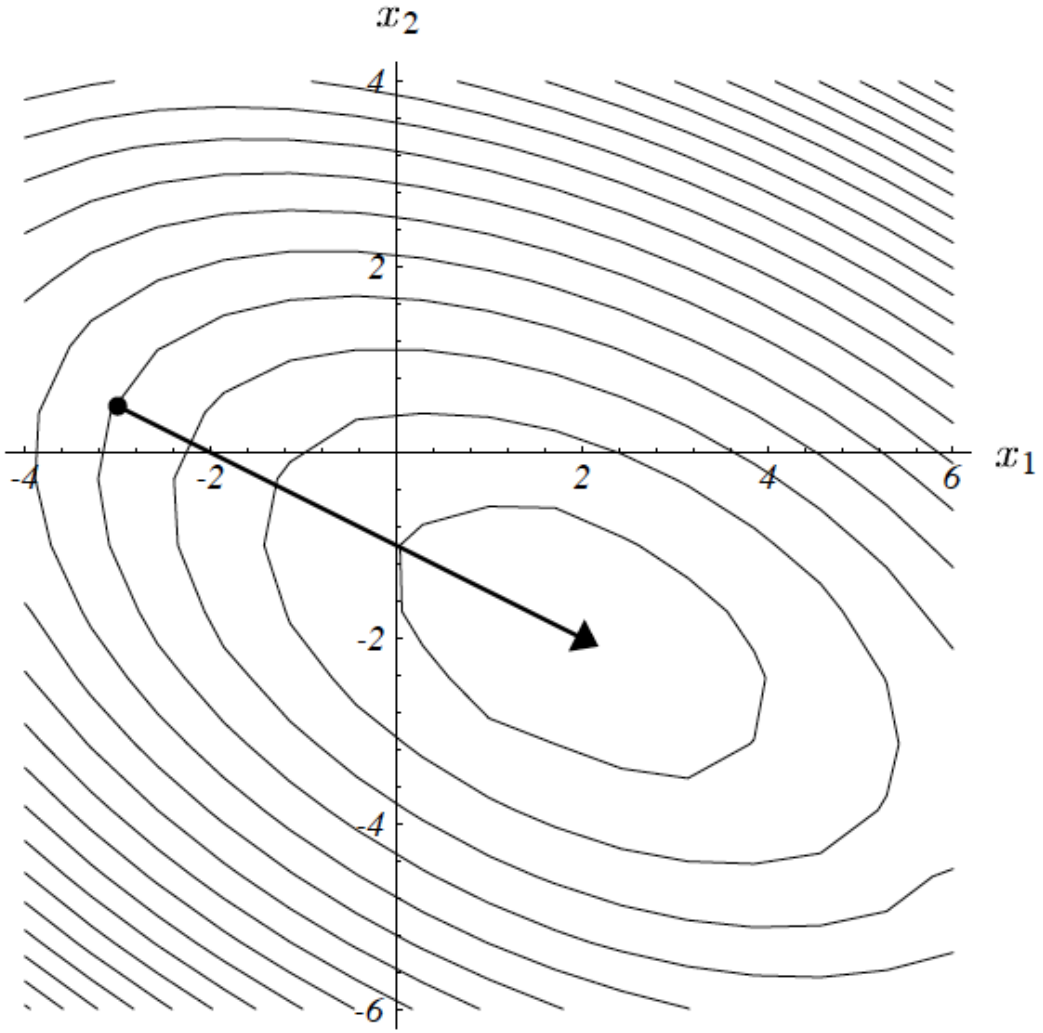
Why is there a distinct zigzag path? Remember that α should be chosen that r_0 and $f'(x_1)$ are orthogonal.

Convergence of Steepest Descent: Instant convergence (cont'd)

There is only one step to the exact solution, because the x_i lies exactly on one axis of the ellipsoid and the residual points directly into the center of the ellipsoid;
thus, $\alpha_i = 1/\lambda_e$ results in instant convergence.

This is a very special case.

For a more general analysis we again think of a vector as a sum of other well-understood vectors i.e. express e_i as a linear combination of *orthonormal* eigenvectors.



Convergence of Steepest Descent: General convergence (cont'd)

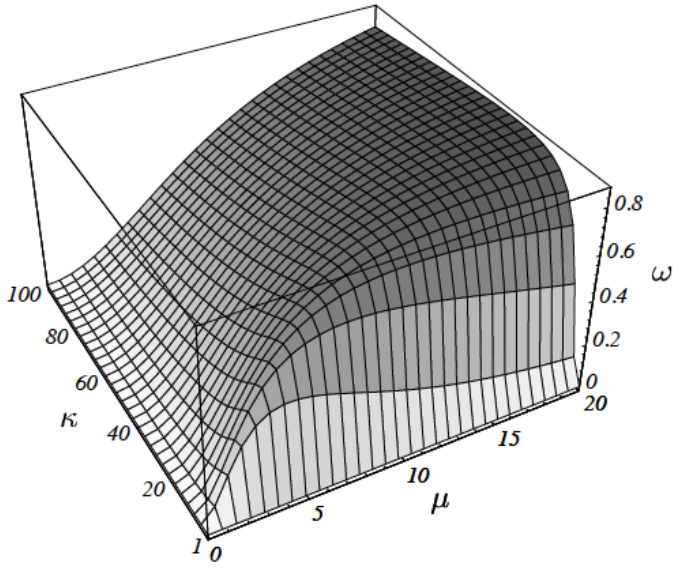
In the suma, for demonstration we have $n = 2$ and assume $\lambda_1 \geq \lambda_2$.

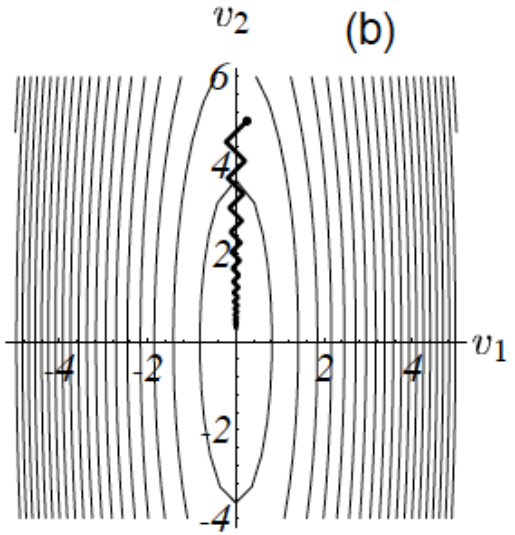
$$\kappa = \frac{\lambda_1}{\lambda_2} \geq 1$$ spectral condition number

$$\mu = \frac{\xi_2}{\xi_1}$$ slope of e_i

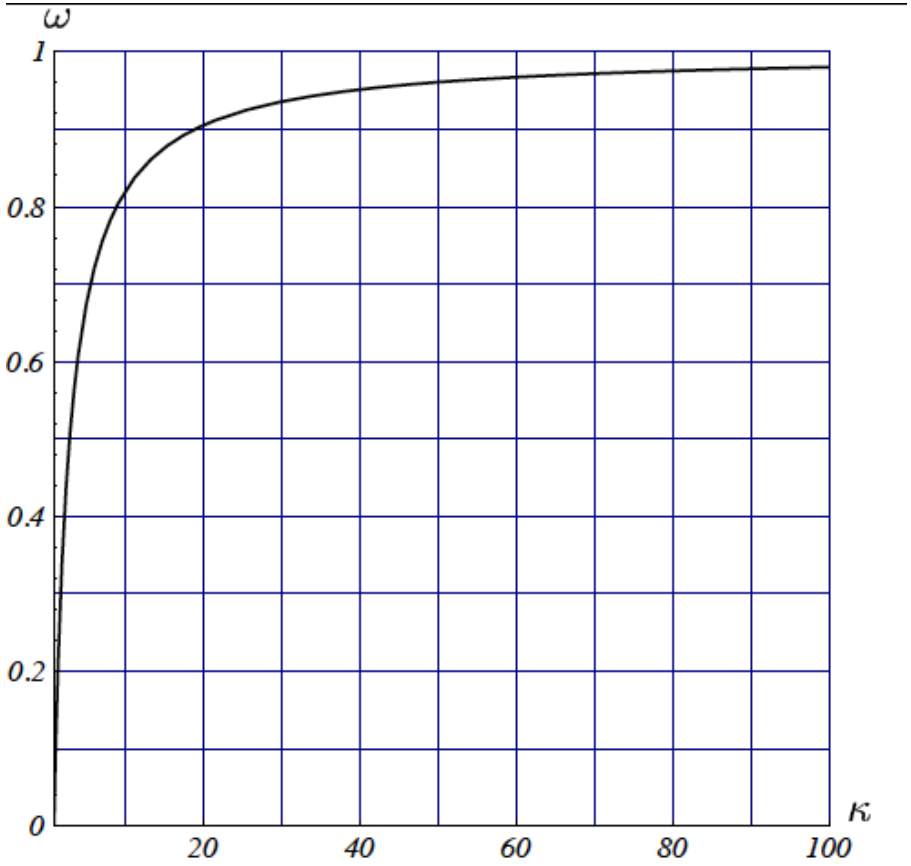
$$\omega^2 = 1 - \frac{\left(\xi_1^2 \lambda_1^2 + \xi_2^2 \lambda_2^2\right)^2}{\left(\xi_1^2 \lambda_1 + \xi_2^2 \lambda_2\right)\left(\xi_1^2 \lambda_1^3 + \xi_2^2 \lambda_2^3\right)}$$

$$= 1 - \frac{\left(\kappa^2 + \mu^2\right)^2}{\left(\kappa + \mu^2\right)\left(\kappa^3 + \mu^2\right)}$$



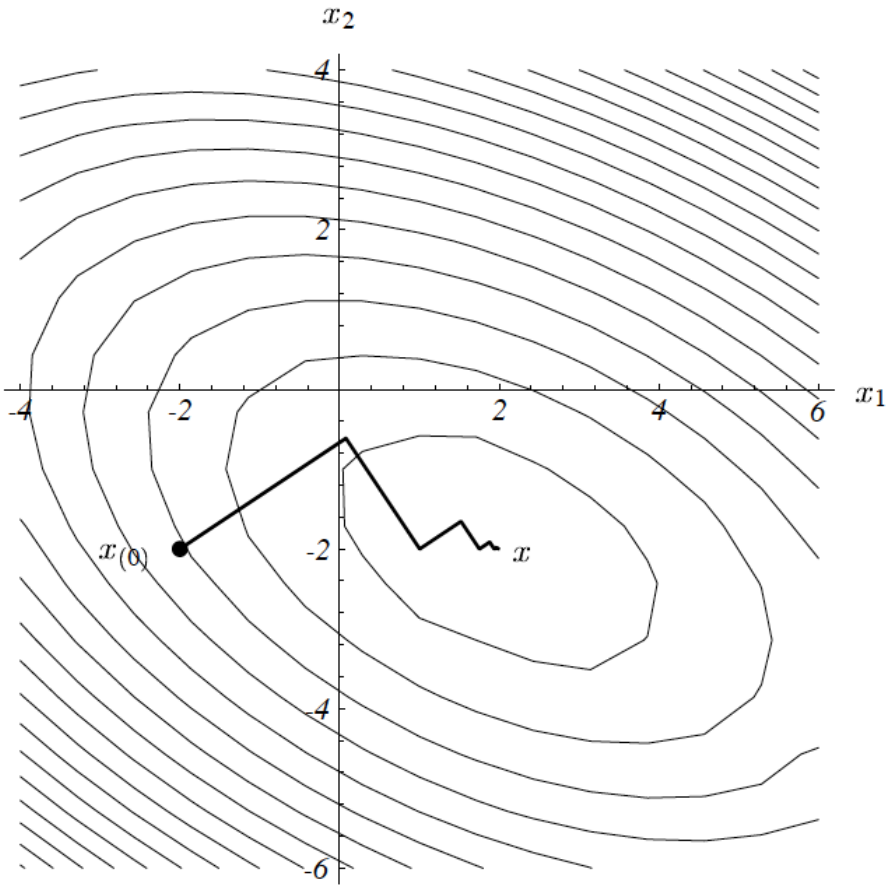


Convergence of Steepest Descent: General convergence (cont'd)



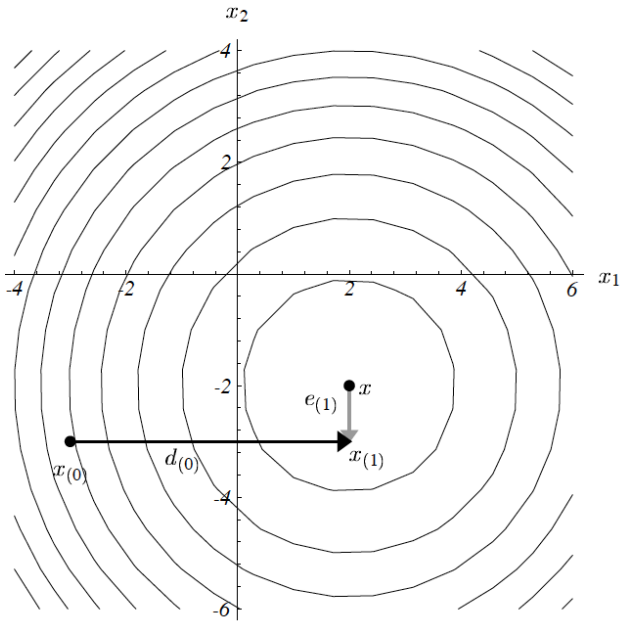
The more ill-conditioned the matrix is i.e. the larger the condition number the slower is the convergence.

What is a disadvantage of Steepest Descent?



Well, there are multiple steps in the same direction of various length.

What if we could find a set of orthogonal search directions d_0, d_1,…,d_n where we take exactly one step in each direction!

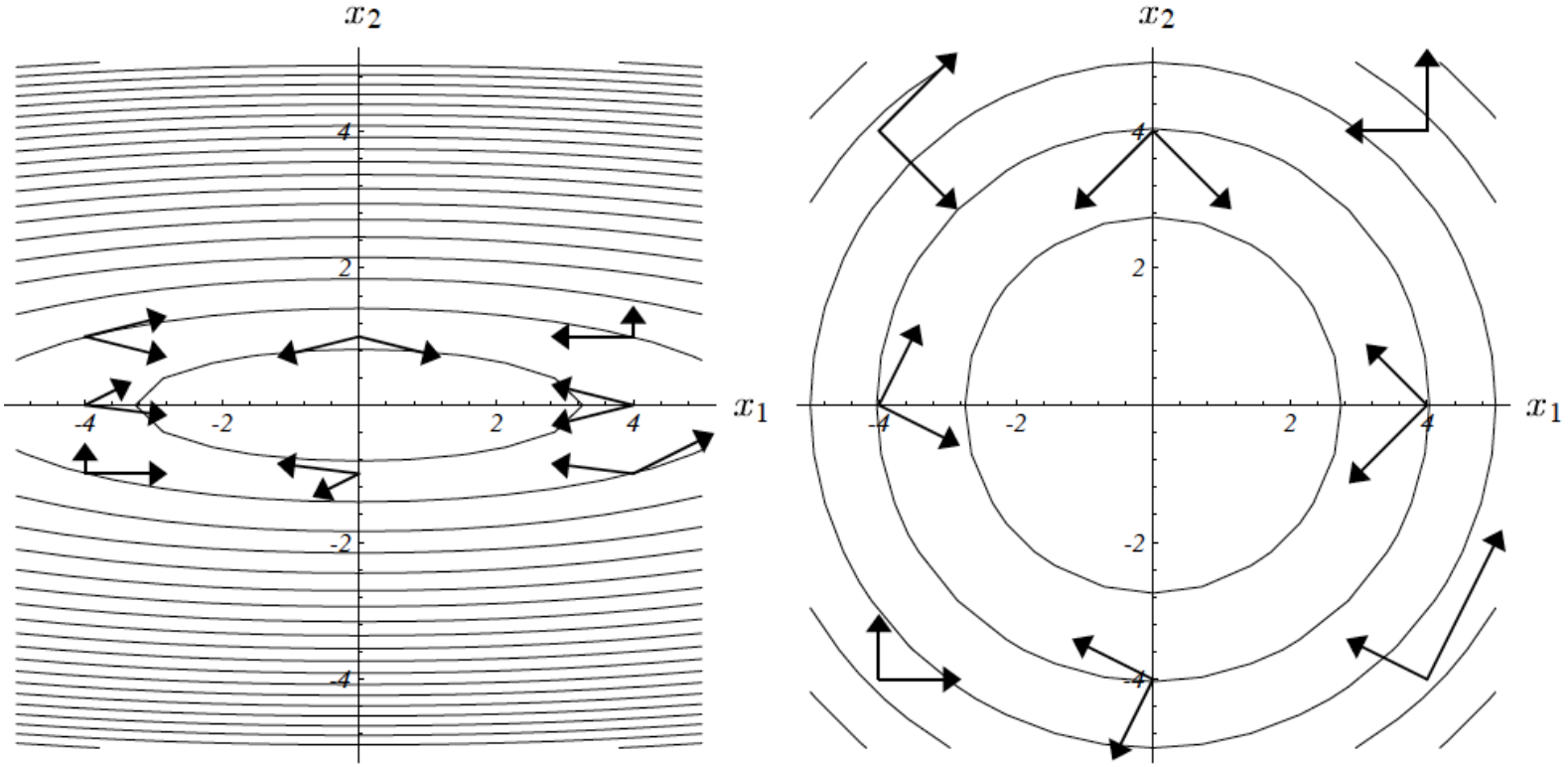


Conjugate Directions (cont'd)

Solution: make the search directions *A*-orthogonal or conjugate instead of orthogonal

$$v_i^T A v_j = 0$$ (here v_i could be the direction, d, and v_j could be some error, e).

These vectors are *A*-orthogonal, because...they are orthogonal after being "stretched".



Gram-Schmidt Conjugation

The last thing needed is a set of A-orthogonal search directions.

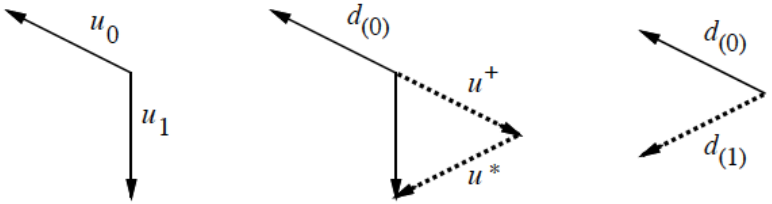
Suppose a set of n linearly independent vectors $u_0, u_1, u_2, \ldots u_n$ (could be the coordinate axes). To construct d_i, take u_i and subtract any components that are not *A*-orthogonal. Thus, set $d_0 = u_0$ and for $i > 0$

$$d_0 = u_0$$

$$d_i = u_i - \sum_{j=0}^{i-1} \beta_{ij} d_j$$

$$\beta_{ij} = \frac{u_i^T A d_j}{d_j^T A d_j}$$

Note, there is an outer loop over $j \leq n$ search directions spanning the whole vector space!



Begin with two linearly independent vectors u_0, u_1. Set $d_0 = u_0$. The vector u_1 is composed of two components: u^* (A-orthogonal to d_0) and u^+ (parallel to d_0). After conjugation only A-orthogonal portion remains and $d_1 = u^*$.

The disadvantage is, all old search vectors must be kept (in memory) to construct the new one and $O(n^3)$ operations are required!

Conjugate gradients (cont'd)

Putting it all together:

$$d_0 = r_0 = b - Ax_0$$

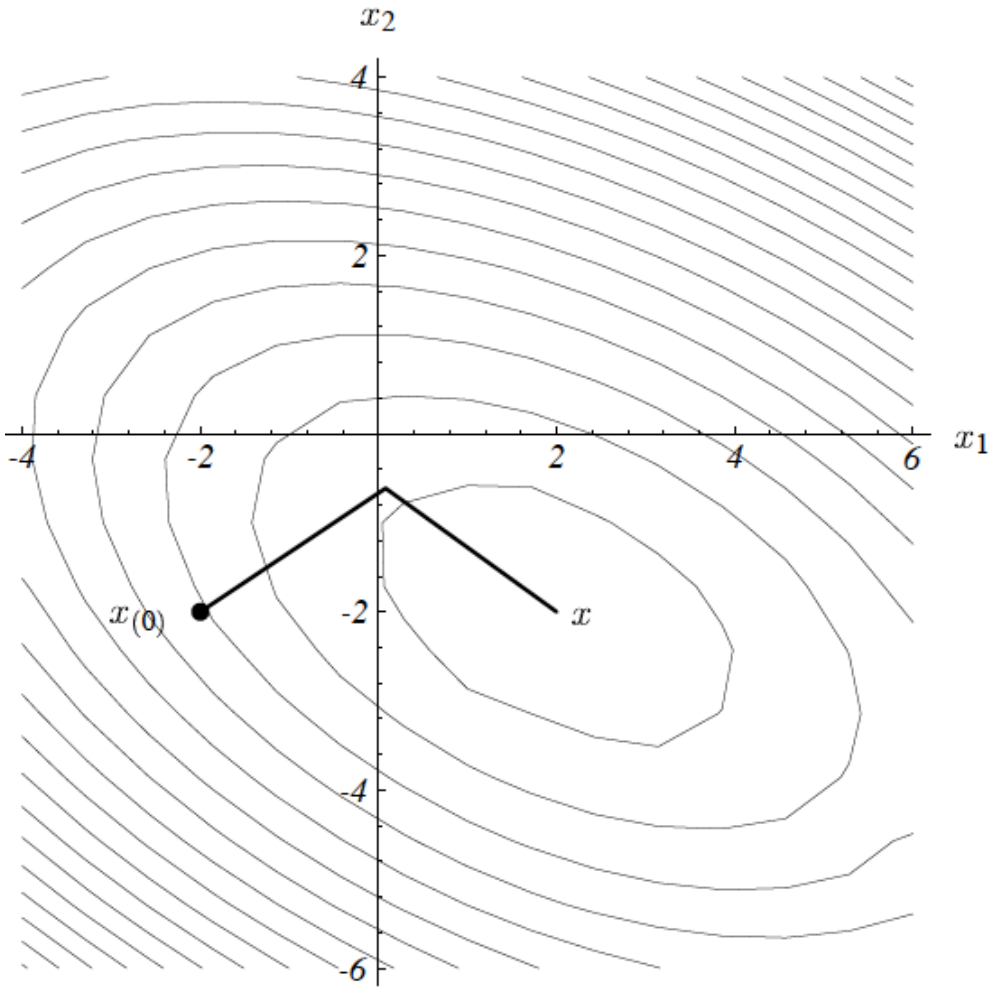
$$\alpha_i = \frac{r_i^T r_i}{d_i^T A d_i}$$

$$x_{i+1} = x_i + \alpha_i d_i$$

$$r_{i+1} = r_i - \alpha_i A d_i$$

$$\beta_{i+1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}$$

$$d_{i+1} = r_{i+1} + \beta_{i+1} d_i$$



Conjugate gradients (cont'd)

Putting it all together:

$$d_0 = r_0 = b - Ax_0$$

$$\alpha_i = \frac{r_i^T r_i}{d_i^T A d_i}$$

$$x_{i+1} = x_i + \alpha_i d_i$$

$$r_{i+1} = r_i - \alpha_i A d_i$$

$$\beta_{i+1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}$$

$$d_{i+1} = r_{i+1} + \beta_{i+1} d_i$$

Given the inputs A, b, x_0, maximum iteration number i_{max}, and error tolerance $\varepsilon < 1$:

$i \Leftarrow 0$
$r \Leftarrow b - Ax$
$d \Leftarrow r$
$\delta_{new} \Leftarrow r^T r$
$\delta_0 \Leftarrow \delta_{new}$
While $i < i_{max}$ and $\delta_{new} > \varepsilon^2 \delta_0$ do
    $q \Leftarrow Ad$
    $\alpha \Leftarrow \frac{\delta_{new}}{d^T q}$
    $x \Leftarrow x + \alpha d$
    If i is divisible by 50
        $r \Leftarrow b - Ax$
    else
        $r \Leftarrow r - \alpha q$
    $\delta_{old} \Leftarrow \delta_{new}$
    $\delta_{new} \Leftarrow r^T r$
    $\beta \Leftarrow \frac{\delta_{new}}{\delta_{old}}$
    $d \Leftarrow r + \beta d$
    $i \Leftarrow i + 1$

Preconditioning (cont'd)

Untransformed CG Method

$$r_0 = b - Ax_0$$

$$d_0 = M^{-1} r_0$$

$$\alpha_i = \frac{r_i^T M^{-1} r_i}{d_i^T A d_i}$$

$$x_{i+1} = x_i + \alpha_i d_i$$

$$r_{i+1} = r_i - \alpha_i A d_i$$

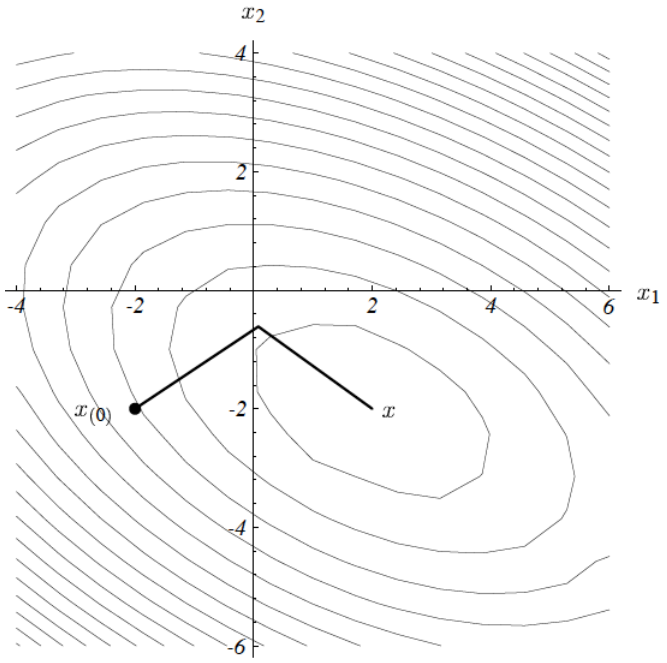
$$\beta_{i+1} = \frac{r_{i+1}^T M^{-1} r_{i+1}}{r_i^T M^{-1} r_i}$$

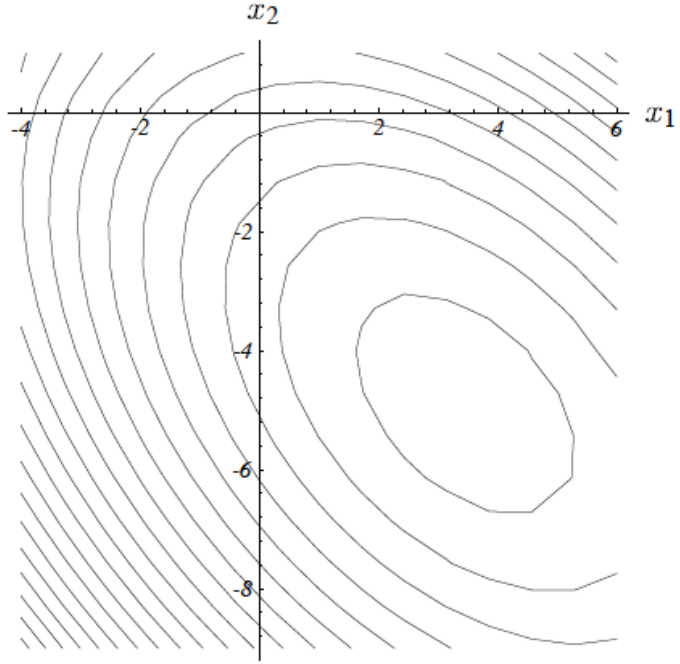
$$d_{i+1} = M^{-1} r_{i+1} + \beta_{i+1} d_i$$

Preconditioning (cont'd)

What does preconditioning due to the quadratic form? It stretched it and makes it appear more spherical.

There is a (very) large number of preconditioners out there and many consider it some kind of "black art". One example is simply a diagonal matrix with entries taken from A, which scales the quadratic form along the coordinate axes. Note, in case of the large problems, application of CG generally always requires a preconditioner of some sorts.





Concrete example of how to solve a set of non-linear system of equations

Additional assumptions:

1. We neglect gravity (horizontal flow only) in 1D
2. $C(p)$ = const. and $D = K / C$ = const.

$$\frac{\partial p_w}{\partial t} = D \frac{\partial}{\partial x}\left[k_w(p_w) \frac{\partial p_w}{\partial x} \right]$$

Finite difference, implicit

$$\frac{p_i^{k+1} - p_i^k}{\Delta t} = \frac{D}{\Delta x^2}\left[k_{uw}(p)\left(p_{i-1}^{k+1} - p_i^{k+1}\right) + k_{uw}(p)\left(p_{i+1}^{k+1} - p_i^{k+1}\right) \right]$$

Instead of using the harmonic mean for the relative permeabilities $k(p)$, we upwind $k_{uw}(p)$ i.e. take the values from the cell where the flow is coming from!

Concrete example and how to solve a set of non-linear equation

We can re-write the equation

$$0 \equiv F(p^k) = p_i^{k+1} - p_i^k - \frac{D\Delta t}{\Delta x^2}\left[k_{uw}(p)\left(p_{i-1}^{k+1} - p_i^{k+1}\right) + k_{uw}(p)\left(p_{i+1}^{k+1} - p_i^{k+1}\right)\right]$$

$$0 \equiv F\left(p^{k+1}\right) = p^{k+1} - p^k - \mathbf{C}f\left(p^{k+1}\right)$$

$\mathbf{C}$ is a matrix containing the constant coefficients, $D\Delta t/\ \Delta x^2$.
$\mathbf{C}$ is not the $\mathbf{A}$ matrix from our $\mathbf{A}\mathrm{x} = \mathrm{b}$ problem!
While $\mathbf{C}$ is a sparse matrix, its shape differs from $\mathbf{A}$.

Applying the Newton-Raphson method

$$0 \equiv F\left(p^{k+1}\right) = p^{k+1} - p^{k} - \mathbf{C}f\left(p^{k+1}\right)$$

Because of the nonlinearity, we only achieve an approximation $\tilde{p}^{k+1}$ of the true p^{k+1} with one linear solve of the above system of equations. But we may be able to improve the approximation in an iterative fashion.

Using (again) a Taylor expansion we have

$$F\left(p^{k+1}\right) \approx F\left(\tilde{p}^{k+1}\right) + F'\left(\tilde{p}^{k+1}\right)\left(p^{k+1} - \tilde{p}^{k+1}\right)$$

An improved approximation of p^{k+1} is obtained by

$$\tilde{p}^{k+1} + d\tilde{p}^{k+1}$$

$$-F'\left(\tilde{p}^{k+1}\right)d\tilde{p}^{k+1} = F\left(\tilde{p}^{k+1}\right)$$

Solve the system of equations to find the update

$$d\tilde{p}^{k+1} = -F'\left(\tilde{p}^{k+1}\right)^{-1} F\left(\tilde{p}^{k+1}\right)$$

update — Jacobi matrix — Approx. solution vector

Applying the Newton-Raphson method (cont'd)

The Jacobian

$$F'\left(p^{k+1}\right) = J\left(p^{k+1}\right) = \frac{\partial\left(p^{k+1} - p^{k}\right)}{\partial p^{k+1}} - \mathbf{C}\frac{\partial f\left(p^{k+1}\right)}{\partial p^{k+1}} = 1 - \mathbf{C}\frac{\partial f\left(p^{k+1}\right)}{\partial p^{k+1}}$$

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial p_1} & \frac{\partial F_1}{\partial p_2} & \cdots & \\ \frac{\partial F_2}{\partial p_1} & \frac{\partial F_2}{\partial p_2} & \frac{\partial F_2}{\partial p_3} & \\ \vdots & & \ddots & \vdots \\ & & \cdots & \frac{\partial F_N}{\partial p_N} \end{bmatrix}$$

What happens if $k(p) = k = \text{const.}$ in $f(p^{k+1})$?

And how does J look in case of 2D and 3D?

Applying the Newton-Raphson method (cont'd)

$$F_1 = p_1^{k+1} - p_1^k - \frac{D\Delta t}{\Delta x^2}\left[k_{uw}(p_1)\left(p_0^{k+1} - p_1\right) + k_{uw}(p_2)\left(p_2^{k+1} - p_1^{k+1}\right)\right]$$

$$\frac{\partial F_1}{\partial p_2} = -\frac{D\Delta t}{\Delta x^2}\left[\frac{\partial k_{uw}(p_2)}{\partial p_2}\left(p_2 - p_1\right) + k_{uw}(p_2)\right]$$

$$F_2 = p_2^{k+1} - p_2^k - \frac{D\Delta t}{\Delta x^2}\left[k_{uw}(p_2)\left(p_1^{k+1} - p_2^{k+1}\right) + k_{uw}(p_3)\left(p_3^{k+1} - p_2^{k+1}\right)\right]$$

$$\frac{\partial F_2}{\partial p_1} = -\frac{D\Delta t}{\Delta x^2}\left[-k_{uw}(p_2)\right]$$

Etc.

Each row contains the transposed gradients of the function f.
Note, the Jacobian can also be approximated via a Taylor expansion (differencing), which however requires a large number of non-linear function evaluations.

Applying the Newton-Raphson method (cont'd)

The different steps of the Newton Raphson are

1. Calculate $F(p^n)$ starting e.g. from an initial guess of p, where n is the number of Newton-Raphson iteration.
2. Calculate the derivative $F'(p^n)$ i.e. the Jacobi matrix
3. The improvement dp^n is calculated as $dp^n = -F'(p^n)^{-1}F$
4. Calculate $p^{n+1} = p^n + dp^n$
5. Repeat steps 1 to 4 until $norm(dp^n)$ has decreases below a pre-defined limit

MATLAB/pseudo-code snipped:

```
niter = 0
while (niter < nmax & err > toll),
  niter = niter+1;
  [relperm,drelperm] = my_relperm(...)
  fp = my_fp()
  F = p – p_old – C*fp; % approx. soln vec
  dF = my_jacobian(…); % Analytical J or FD
  dp = -dF\F; %Solve for the update dp
  err = abs(dp); % Is this correct?
  p = p+dp
end
```

Thus, the Newton-Raphson method combines an outer non-linear iteration method with an inner linear systems solver approach.

The basic approach makes up the large class of Newton-PCG solvers, where a Newton method is combined with a PCG linear solver.