Alternative methods for solving power flow problems

Tomas Tinoco De Rubira

Stanford University, Stanford, CA

Electric Power Research Institute, Palo Alto, CA

October 25, 2012

Outline

1. **introduction**
2. **available methods**
3. **contributions and proposed methods**
4. **implementation**
5. **experiments**
6. **conclusions**

1. Introduction

1.1. **background**

1.2. **formulation**

1.1. background

1.1. background

power flow problem

1.1. background

power flow problem

- (roughly) given gen and load powers, find [5][6]
  - voltage magnitudes and angles at every bus
  - real and reactive power flows through every branch

1.1. background

power flow problem

- (roughly) given gen and load powers, find [5][6]
  - voltage magnitudes and angles at every bus
  - real and reactive power flows through every branch

- essential for:
  - expansion, planning and daily operation [5]
  - real time security analysis (contingencies) [44]

1.1. background

power flow problem

- (roughly) given gen and load powers, find [5][6]
  - voltage magnitudes and angles at every bus
  - real and reactive power flows through every branch

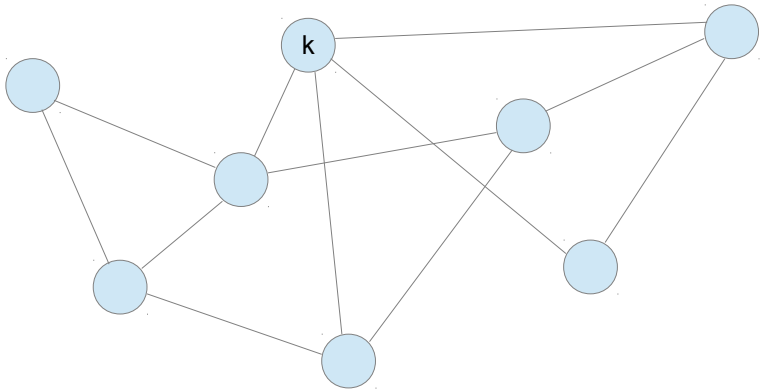
- essential for:
  - expansion, planning and daily operation [5]
  - real time security analysis (contingencies) [44]

- fast and reliable solution methods needed

1.2. formulation

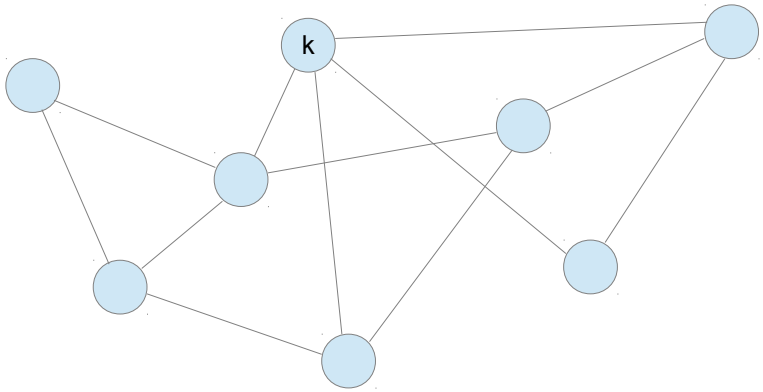
1.2. formulation

network



1.2. formulation

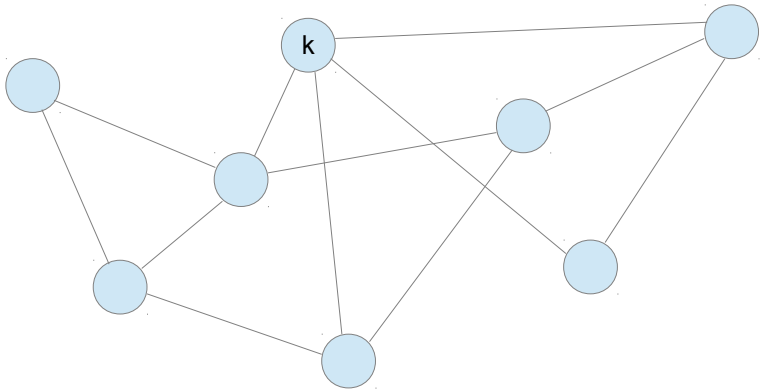
network



flow in $-$ flow out $= 0$ at each bus

1.2. formulation

network



flow in $-$ flow out $= 0$ at each bus

gives power flow equations [5][6]

$$\Delta P_k = P_k^g - P_k^l - \sum_{m \in [n]} v_k v_m (G_{km} \cos(\theta_k - \theta_m) + B_{km} \sin(\theta_k - \theta_m)) = 0$$

$$\Delta Q_k = Q_k^g - Q_k^l - \sum_{m \in [n]} v_k v_m (G_{km} \sin(\theta_k - \theta_m) - B_{km} \cos(\theta_k - \theta_m)) = 0$$

typically: [5][6]

typically: [5][6]

partition $[n] = \{s\} \cup \mathcal{R} \cup \mathcal{U}$

typically: [5][6]

partition $[n] = \{s\} \cup \mathcal{R} \cup \mathcal{U}$

- bus $i = s$: slack
  - voltage magnitude and angle given

typically: [5][6]

partition $[n] = \{s\} \cup \mathcal{R} \cup \mathcal{U}$

- bus $i = s$: slack
  - voltage magnitude and angle given

- bus $i \in \mathcal{R}$: regulated
  - active power and voltage magnitude given (PV)

    (target magnitude v_i^t)

typically: [5][6]

partition $[n] = \{s\} \cup \mathcal{R} \cup \mathcal{U}$

- bus $i = s$: slack
  - voltage magnitude and angle given

- bus $i \in \mathcal{R}$: regulated
  - active power and voltage magnitude given (PV)

    (target magnitude v_i^t)

- bus $i \in \mathcal{U}$: unregulated
  - active and reactive power given (PQ)

2. Available methods

2.1. **overview**

2.2. **Newton-Raphson**

2.3. **Newton-Krylov**

2.4. **optimal multiplier**

2.5. **continuous Newton**

2.6. **sequential conic programming**

2.7. **holomorphic embedding**

2.1. overview

2.1. overview

- mid 1950s, Gauss-Seidel method [47]
  - low memory requirements
  - lacked robustness and good convergence properties
  - worse as network sizes ↑

2.1. overview

- mid 1950s, Gauss-Seidel method [47]
  - low memory requirements
  - lacked robustness and good convergence properties
  - worse as network sizes ↑

- Newton-Raphson (NR) method [5][47]
  - more robust, better convergence properties
  - good when combined with sparse techniques
  - most widely used method

- improving NR
  - speed: fast decoupled [5], DC power flow, simplified XB and BX [47]
  - scalability: Newton-Krylov [13][14][15][20][26][32][35][45]
  - robustness: integration [34] and optimal multiplier [9][11][27][41][42]

- improving NR
  - speed: fast decoupled [5], DC power flow, simplified XB and BX [47]
  - scalability: Newton-Krylov [13][14][15][20][26][32][35][45]
  - robustness: integration [34] and optimal multiplier [9][11][27][41][42]

- other methods
  - genetic, neural networks and fuzzy algorithms
    * not competitive with NR-based method [44]
  - sequential conic programming [28][29]
  - holomorphic embedding [43]

2.2. Newton-Raphson

2.2. Newton-Raphson

iterative method for solving $f(x) = 0$, f is C^1

2.2. Newton-Raphson

iterative method for solving $f(x) = 0$, f is C^1

- linearize at x_k: $f_k + J_k(x_{k+1} - x_k) = 0$
- update: $x_{k+1} = x_k + p_k$, where $J_k p_k = -f_k$
- stop when $\|f_k\|_\infty < \epsilon$

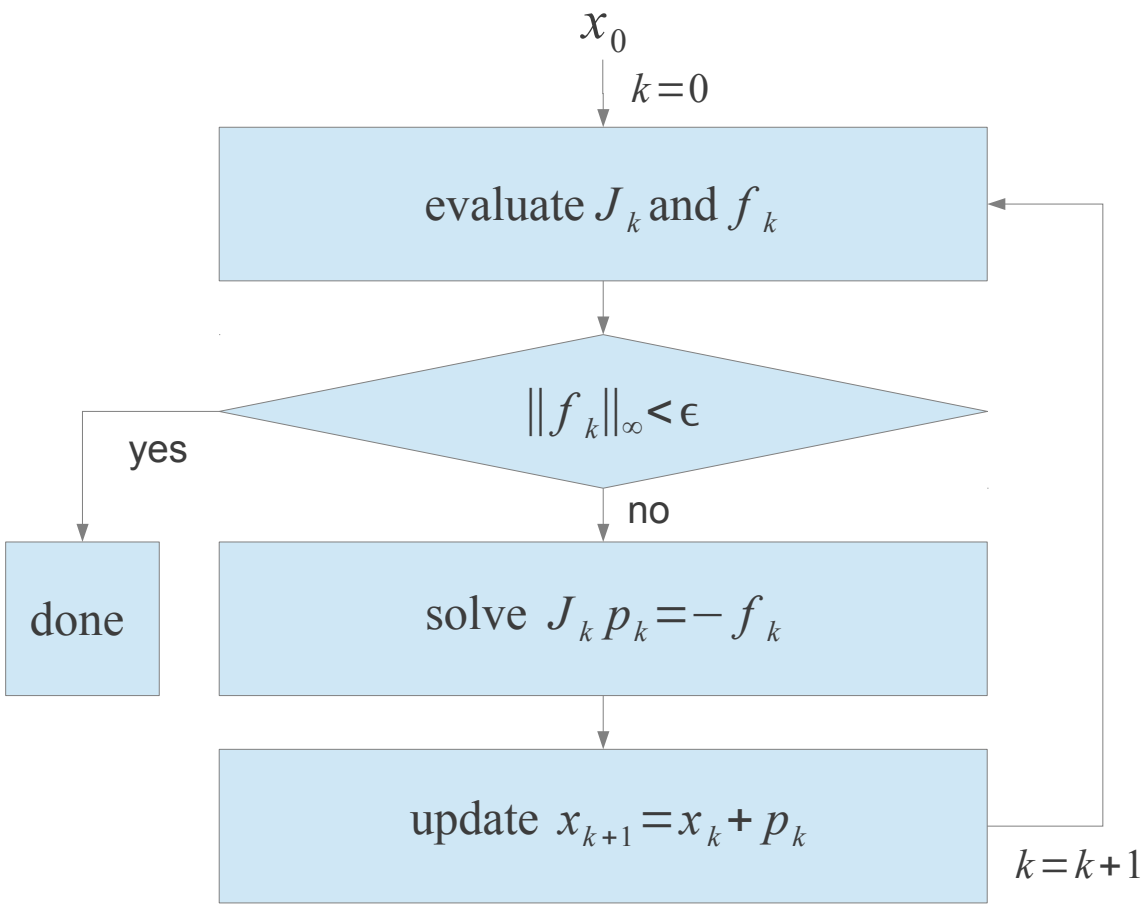
2.2. Newton-Raphson

iterative method for solving $f(x) = 0$, f is C^1

- linearize at x_k: $f_k + J_k(x_{k+1} - x_k) = 0$
- update: $x_{k+1} = x_k + p_k$, where $J_k p_k = -f_k$
- stop when $\|f_k\|_\infty < \epsilon$

properties: [36]

- quadratic rate of convergence
- only locally convergent


x_0
$k=0$
evaluate J_k and f_k
$\|f_k\|_\infty < \epsilon$
yes
no
done
solve $J_k p_k = -f_k$
update $x_{k+1} = x_k + p_k$
$k=k+1$

for **power flow**: [5][6]

for **power flow**: [5][6]

- f is "power mismatches", left hand side of

$$\Delta P_i = 0, \quad i \in \mathcal{R} \cup \mathcal{U}$$
$$\Delta Q_i = 0, \quad i \in \mathcal{U}$$

- x is voltage magnitudes $\{v_i\}_{i \in \mathcal{U}}$ and angles $\{\theta_i\}_{i \in [n] \setminus \{s\}}$
- other variables updated separately to enforce rest of equations
- heuristics to enforce $Q_i^{\min} \leq Q_i^g \leq Q_i^{\max}$, $i \in \mathcal{R}$

for **power flow**: [5][6]

- f is "power mismatches", left hand side of

$$\Delta P_i = 0, \quad i \in \mathcal{R} \cup \mathcal{U}$$
$$\Delta Q_i = 0, \quad i \in \mathcal{U}$$

- x is voltage magnitudes $\{v_i\}_{i \in \mathcal{U}}$ and angles $\{\theta_i\}_{i \in [n] \setminus \{s\}}$
- other variables updated separately to enforce rest of equations
- heuristics to enforce $Q_i^{\min} \leq Q_i^g \leq Q_i^{\max}$, $i \in \mathcal{R}$
  - If violation
    - $Q_i^g \rightarrow$ fixed
    - $\mathcal{R} \rightarrow \mathcal{U}$
    - $v_i \rightarrow$ free
  - more complex ones for $\mathcal{R} \leftarrow \mathcal{U}$ [46]

2.3. Newton-Krylov

2.3. Newton-Krylov

size of networks ↑, $\boxed{J_k p_k = -f_k \text{ problematic for direct methods}}$ [45]

2.3. Newton-Krylov

size of networks ↑, $\boxed{J_k p_k = -f_k \text{ problematic for direct methods}}$ [45]

Krylov subspace methods [40]

- GMRES [13][15][20][26], FGMRES [45]
- BCGStab [20][32][35]
- CGS [20]

2.3. Newton-Krylov

size of networks ↑, $\boxed{J_k p_k = -f_k \text{ problematic for direct methods}}$ [45]

Krylov subspace methods [40]

- GMRES [13][15][20][26], FGMRES [45]
- BCGStab [20][32][35]
- CGS [20]

preconditioners

- polynomial [14][32]
- LU-based [20][26][35][45]

J_k changes little $\rightarrow$ same preconditioner, multiple iterations [15][20][26]

2.4. optimal multiplier

2.4. optimal multiplier

NR only locally convergent [36], can even diverge

2.4. optimal multiplier

NR only locally convergent [36], can even diverge

ensure progress with controlled update: $x_{k+1} = x_k + \alpha_k p_k$

- α_k minimizes $\|f(x_k + \alpha_k p_k)\|_2^2$
- easy in rectangular coordinates [27]
  - find roots of cubic polynomial

2.4. optimal multiplier

NR only locally convergent [36], can even diverge

ensure progress with controlled update: $x_{k+1} = x_k + \alpha_k p_k$

- α_k minimizes $\|f(x_k + \alpha_k p_k)\|_2^2$
- easy in rectangular coordinates [27]
  - find roots of cubic polynomial

extend to polar [9][11][27][41]

- polar sometimes preferable [42]
- can only minimize approx. of $\|f(x_k + \alpha_k p_k)\|_2^2$, no guarantees

2.5. continuous Newton

2.5. continuous Newton

cast power flow as ODE $\rightarrow$ apply robust integration techniques [34]

2.5. continuous Newton

cast power flow as ODE $\rightarrow$ apply robust integration techniques [34]

- NR method $\leftrightarrow$ forward Euler with $\Delta t = 1$
- optimal multiplier [27] $\leftrightarrow$ a type of variable-step forward Euler

2.5. continuous Newton

cast power flow as ODE $\rightarrow$ apply robust integration techniques [34]

- NR method $\leftrightarrow$ forward Euler with $\Delta t = 1$
- optimal multiplier [27] $\leftrightarrow$ a type of variable-step forward Euler

Runge-Kutta [34]

- more robust than NR
- less iterations than the optimal multiplier method (UCTE network)
- no speed comparison

2.6. sequential conic programming

2.6. sequential conic programming

for radial networks [28]

- change variables, cast power flow problem as conic program
  - convex, easy to solve (in theory)

2.6. sequential conic programming

for radial networks [28]

- change variables, cast power flow problem as conic program
  - convex, easy to solve (in theory)

for meshed networks [29] (same author)

- extra constraints: zero angle changes around every loop
  - destroy convexity
- linearize and solve sequence of conic programs (expensive)

relatively large number of iterations [29]

2.7. holomorphic embedding

2.7. holomorphic embedding

embed power flow equations in system of eqs. with extra $s \in \mathbb{C}$ [43]

- $s = 0$ (zero net injections), $s = 1$ (original)
- complex voltage solutions $\tilde{v}(s)$ are holomorphic functions of s

2.7. holomorphic embedding

embed power flow equations in system of eqs. with extra $s \in \mathbb{C}$ [43]

- $s = 0$ (zero net injections), $s = 1$ (original)
- complex voltage solutions $\tilde{v}(s)$ are holomorphic functions of s

find power series of $\tilde{v}(s)$ at $s = 0$ (solve linear system)

apply analytic continuation (Padé approximants [7]) to get $\tilde{v}(s)$ at $s = 1$

2.7. holomorphic embedding

embed power flow equations in system of eqs. with extra $s \in \mathbb{C}$ [43]

- $s = 0$ (zero net injections), $s = 1$ (original)
- complex voltage solutions $\tilde{v}(s)$ are holomorphic functions of s

find power series of $\tilde{v}(s)$ at $s = 0$ (solve linear system)

apply analytic continuation (Padé approximants [7]) to get $\tilde{v}(s)$ at $s = 1$

very strong claims in [43]:

- algorithm finds solution iff "solution" exists

few experimental results

3. Contributions and proposed methods

3.1. **line search**

3.2. **starting point**

3.3. **formulation and bus type switching**

3.4. **real networks**

3.1. line search

3.1. line search

choose α_k for $x_{k+1} = x_k + \alpha_k p_k$ (common in optimization)

- can ensure progress & make NR globally convergent [36]

3.1. line search

choose α_k for $x_{k+1} = x_k + \alpha_k p_k$ (common in optimization)

- can ensure progress & make NR globally convergent [36]

most NR-based methods reviewed: no line search

3.1. line search

choose α_k for $x_{k+1} = x_k + \alpha_k p_k$ (common in optimization)

- can ensure progress & make NR globally convergent [36]

most NR-based methods reviewed: no line search

suggest: should be standard practice

3.1. line search

choose α_k for $x_{k+1} = x_k + \alpha_k p_k$ (common in optimization)

- can ensure progress & make NR globally convergent [36]

most NR-based methods reviewed: no line search

suggest: should be standard practice

- no need for integration [34]
- or approx. rectangular optimal multiplier [9][11][41][27]
- simple bracketing and bisection [38]

3.1. line search

choose α_k for $x_{k+1} = x_k + \alpha_k p_k$ (common in optimization)

- can ensure progress & make NR globally convergent [36]

most NR-based methods reviewed: no line search

suggest: should be standard practice

- no need for integration [34]
- or approx. rectangular optimal multiplier [9][11][41][27]
- simple bracketing and bisection [38]

method: Line Search Newton-Raphson (LSNR)

line search Newton-Raphson (LSNR)

line search Newton-Raphson (LSNR)

formulate as in NR, $f(x) = 0$

measure progress with $h = \frac{1}{2} f^T f$ (merit function)

use controlled update $x_{k+1} = x_k + \alpha_k p_k$

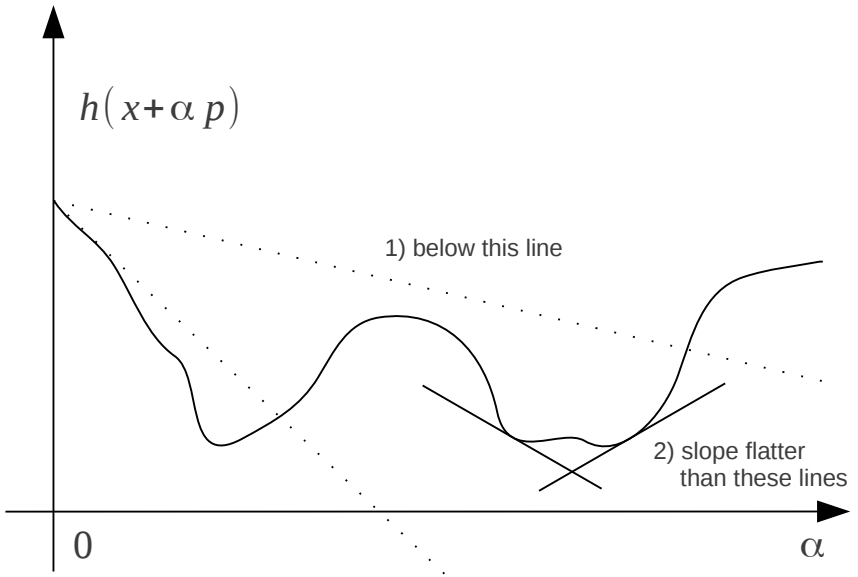
line search Newton-Raphson (LSNR)

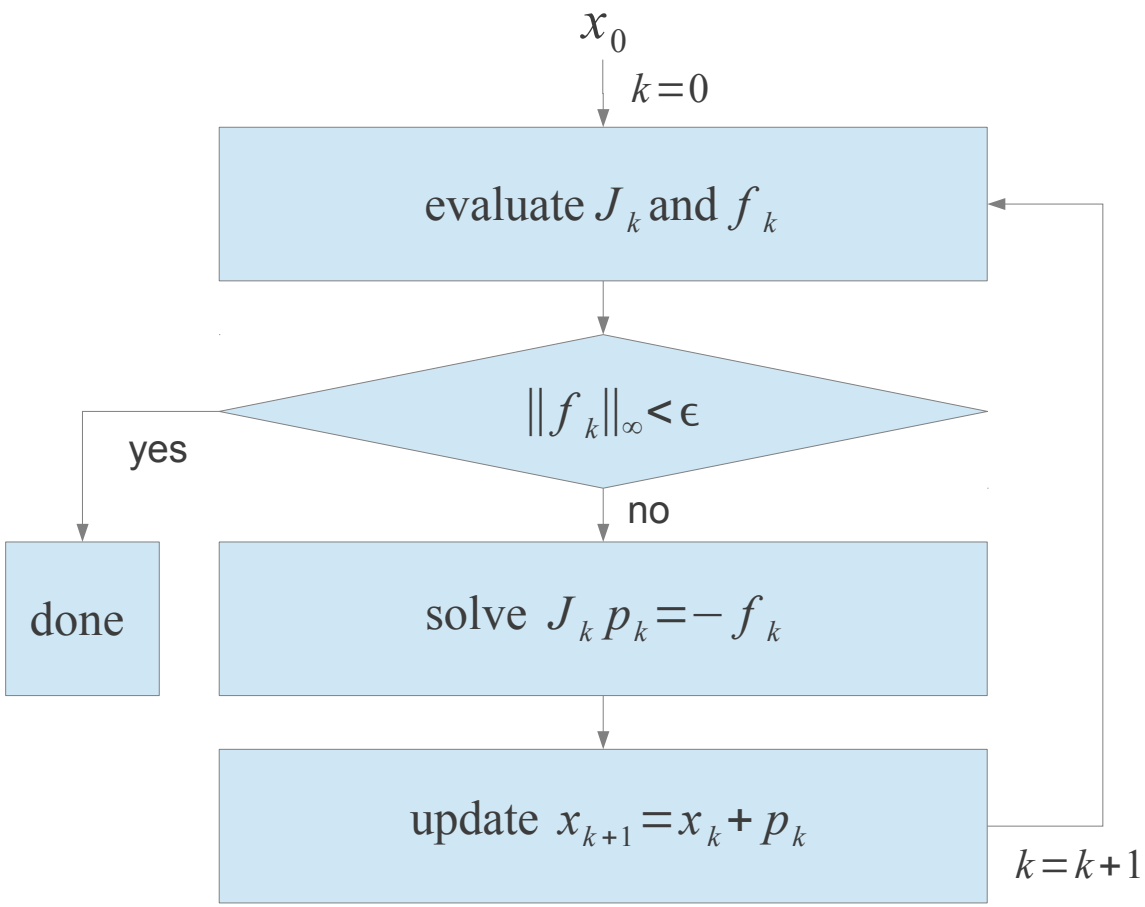
formulate as in NR, $f(x) = 0$

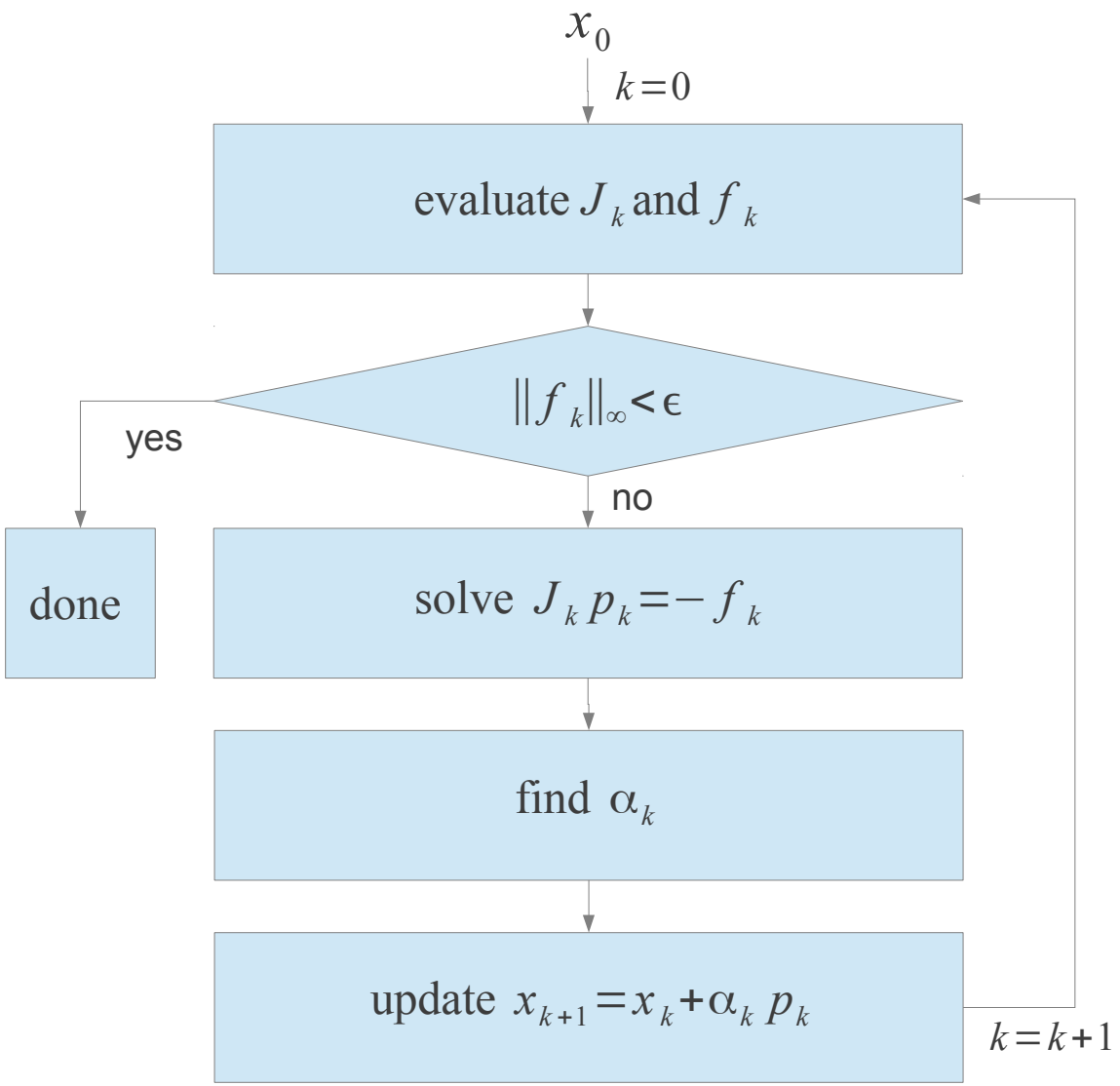
measure progress with $h = \frac{1}{2} f^T f$ (merit function)

use controlled update $x_{k+1} = x_k + \alpha_k p_k$

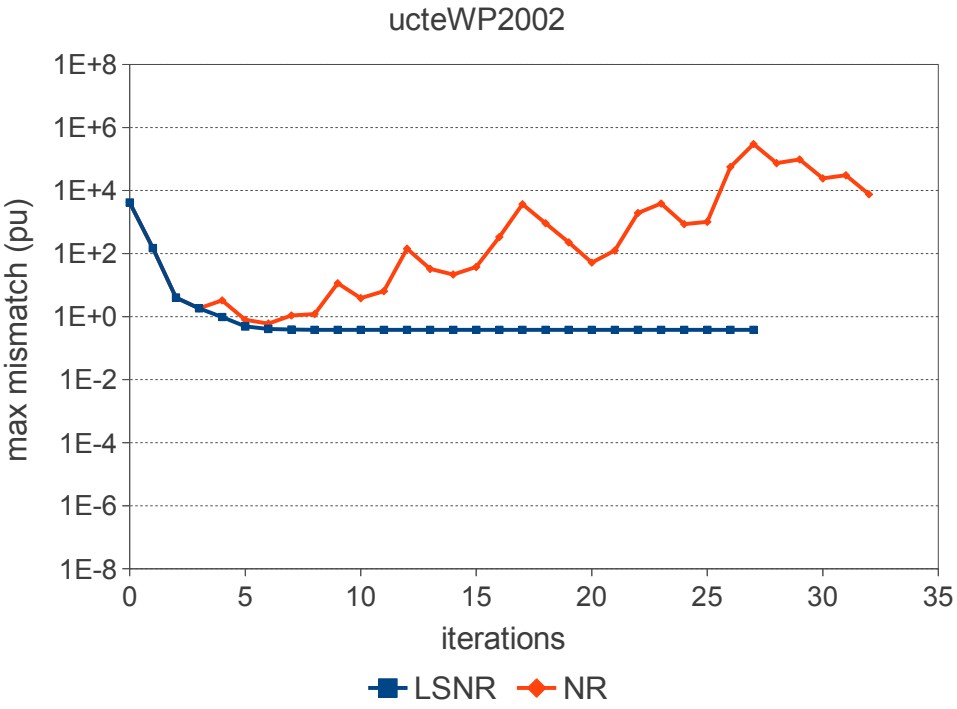
- α_k satisfies strong Wolfe conditions [38]

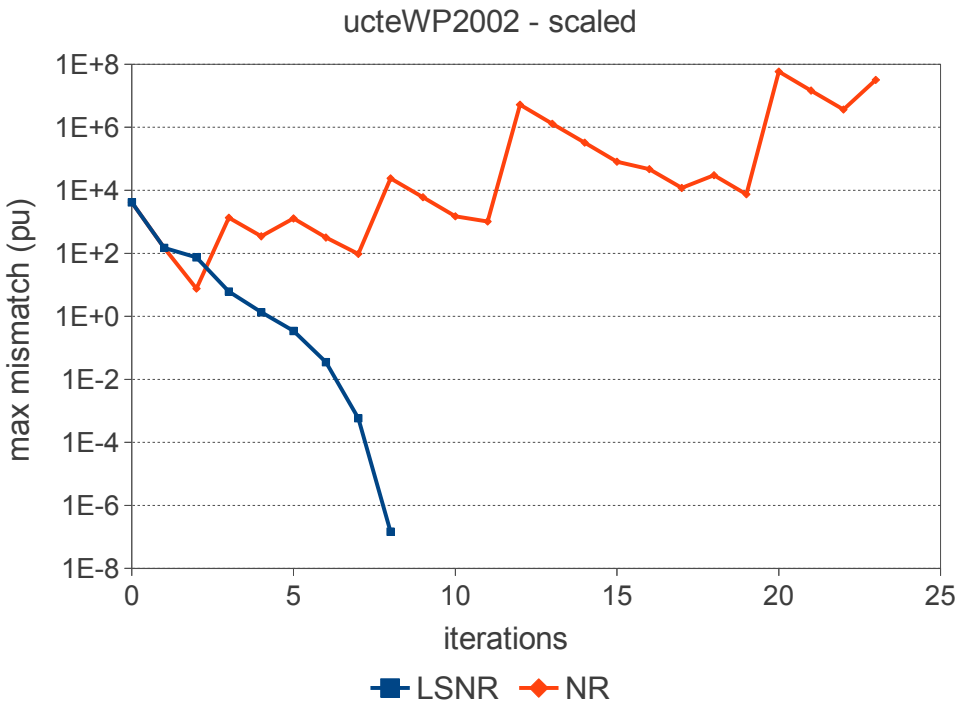



x_0
$k=0$
evaluate J_k and f_k
$\|f_k\|_\infty < \epsilon$
yes
no
done
solve $J_k p_k = -f_k$
update $x_{k+1} = x_k + p_k$
$k=k+1$


x_0
$k=0$
evaluate J_k and f_k
$\|f_k\|_\infty < \epsilon$
yes
no
done
solve $J_k p_k = -f_k$
find α_k
update $x_{k+1} = x_k + \alpha_k p_k$
$k=k+1$

benefits





scaling: reduce net injection by factor of 1.07 approx.

3.2. starting point

3.2. starting point

NR needs good x_0

3.2. starting point

NR needs good x_0

often flat start used but can also fail, then

- trial and error: tweak network & guess other x_0

3.2. starting point

NR needs good x_0

often flat start used but can also fail, then

- trial and error: tweak network & guess other x_0

propose: automate transformations (homotopy [31])

3.2. starting point

NR needs good x_0

often flat start used but can also fail, then

- trial and error: tweak network & guess other x_0

propose: automate transformations (homotopy [31])

- transform network to flat network
- gradually transform back
- solutions for transformed networks $\rightarrow$ solution for original network

3.2. starting point

NR needs good x_0

often flat start used but can also fail, then

- trial and error: tweak network & guess other x_0

propose: automate transformations (homotopy [31])

- transform network to flat network
- gradually transform back
- solutions for transformed networks $\rightarrow$ solution for original network

method: Flat Network Homotopy (FNH) (phase shifters only)

flat network homotopy (FNH)

flat network homotopy (FNH)

formulate as $f(x,t) = 0$ (x and f as before)

t scales phase shifters:

- $t = 0$ phase shifters removed (flat network)
- $t = 1$ phase shifters in original state

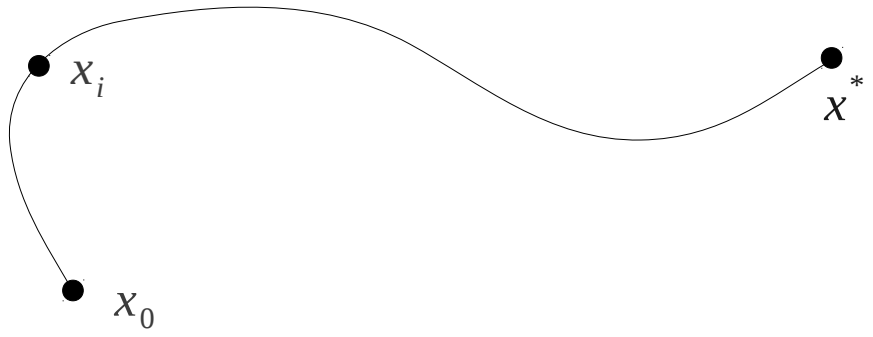
flat network homotopy (FNH)

formulate as $f(x,t) = 0$ (x and f as before)

t scales phase shifters:

- $t = 0$ phase shifters removed (flat network)
- $t = 1$ phase shifters in original state

approx. solve $f(x,t_i) = 0$ for $i \in \mathbb{N}$, $t_1 = 0$, $t_i \uparrow 1$ as $i \to \infty$



how?

how?

approx. solve sequence of problems

$$\underset{x}{\text{minimize}} \quad F_i(x) = \frac{1}{2}\eta_i \sum_{j \in \mathcal{U}} (v_j - 1)^2 + \frac{1}{2}\|f(x, t_i)\|_2^2$$

indexed by i, generate approx. solutions $\{x_i\}_{i \in \mathbb{N}}$

how?

approx. solve sequence of problems

$$\underset{x}{\text{minimize}} \quad F_i(x) = \frac{1}{2}\eta_i \sum_{j \in \mathcal{U}} (v_j - 1)^2 + \frac{1}{2}\|f(x, t_i)\|_2^2$$

indexed by i, generate approx. solutions $\{x_i\}_{i \in \mathbb{N}}$

first term encourages trajectory of good points

- goes away: $\eta_i \downarrow 0$ as $i \to \infty$

how?

approx. solve sequence of problems

$$\underset{x}{\text{minimize}} \quad F_i(x) = \frac{1}{2}\eta_i \sum_{j \in \mathcal{U}} (v_j - 1)^2 + \frac{1}{2}\|f(x, t_i)\|_2^2$$

indexed by i, generate approx. solutions $\{x_i\}_{i \in \mathbb{N}}$

first term encourages trajectory of good points

- goes away: $\eta_i \downarrow 0$ as $i \to \infty$

starting points:

- subproblem $i = 1$ (flat network), use flat start
- subproblem $i > 1$, modify x_{i-1}

3.3. formulation and bus type switching

3.3. formulation and bus type switching

square formulation

- p_k comes from f only: missing info (physical limits & preferences)
- can lead to regions with poor or meaningless points

3.3. formulation and bus type switching

square formulation

- p_k comes from f only: missing info (physical limits & preferences)
- can lead to regions with poor or meaningless points

switching heuristics (PV - PQ)

- keep formulation square
- can cause "jolts", cycling and prevent convergence [46]

3.3. formulation and bus type switching

square formulation

- p_k comes from f only: missing info (physical limits & preferences)
- can lead to regions with poor or meaningless points

switching heuristics (PV - PQ)

- keep formulation square
- can cause "jolts", cycling and prevent convergence [46]

propose: add degrees of freedom, add missing info

3.3. formulation and bus type switching

square formulation

- p_k comes from f only: missing info (physical limits & preferences)
- can lead to regions with poor or meaningless points

switching heuristics (PV - PQ)

- keep formulation square
- can cause "jolts", cycling and prevent convergence [46]

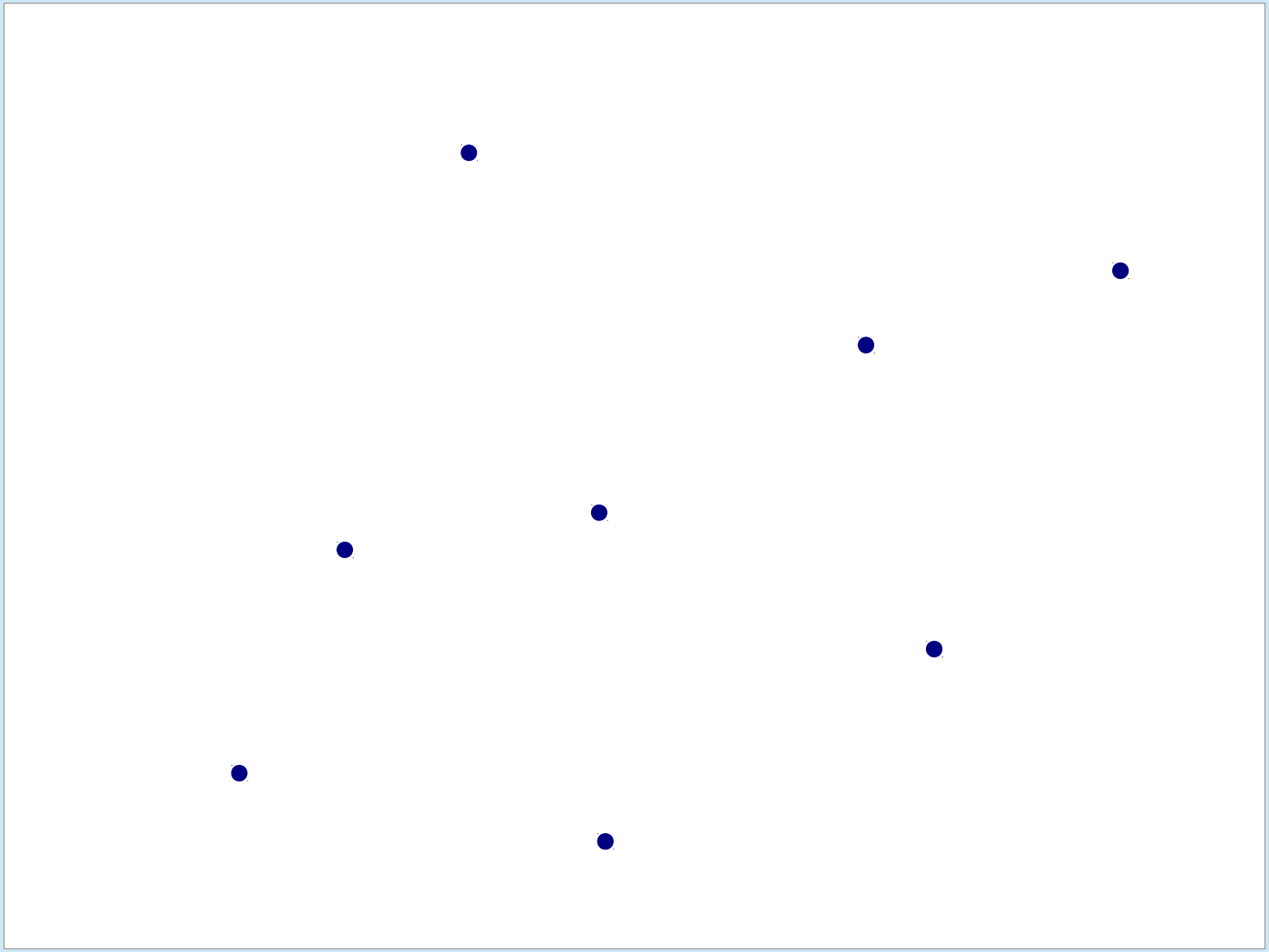
propose: add degrees of freedom, add missing info

methods: Vanishing Regulation (VR), Penalty-Based Power Flow (PBPF)

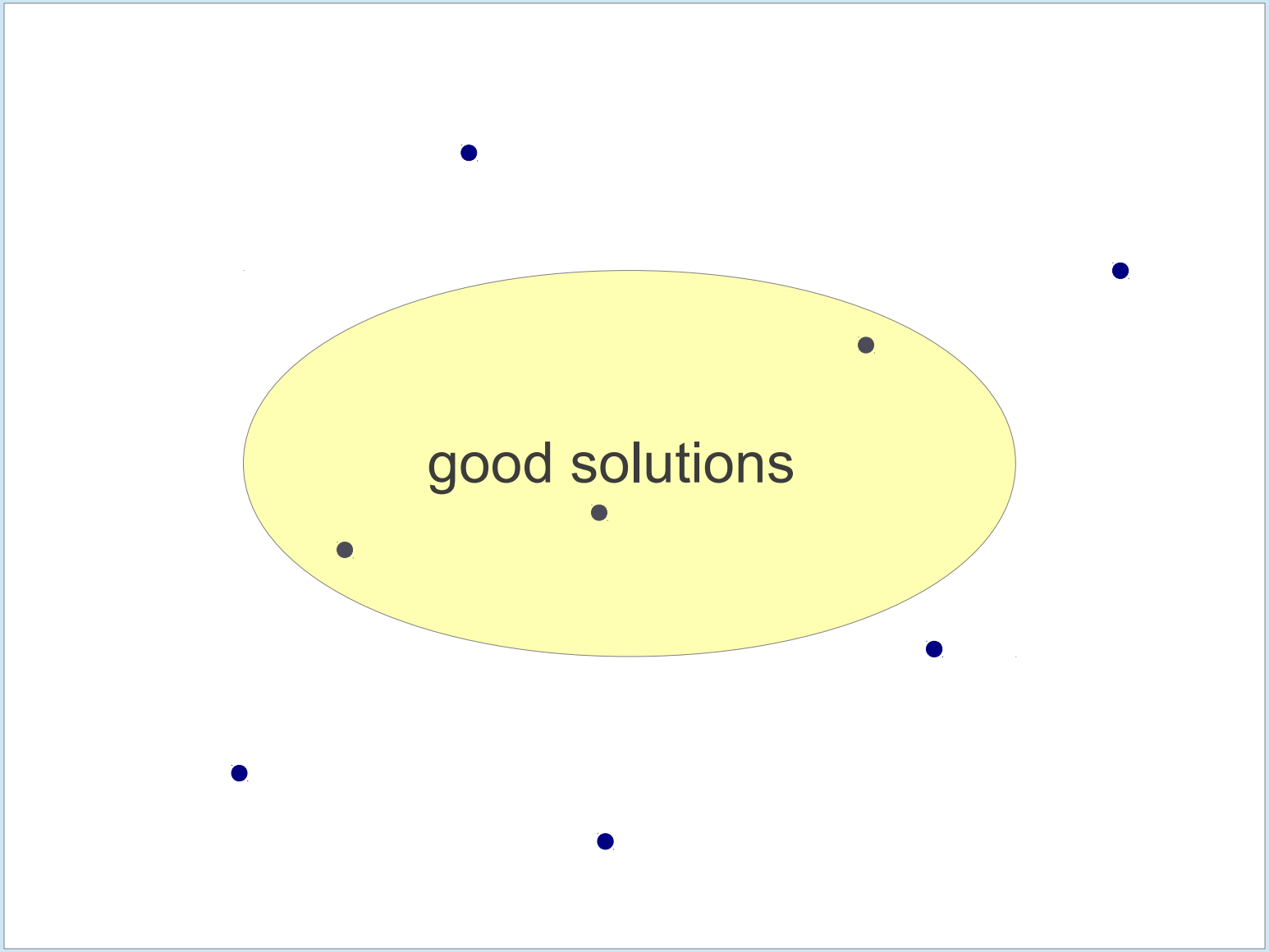
analogy ...

Q violations

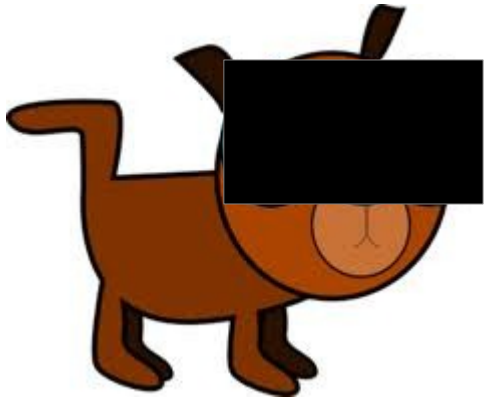
Q violations



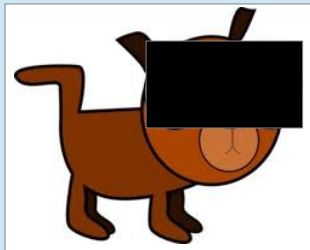
power flow solutions

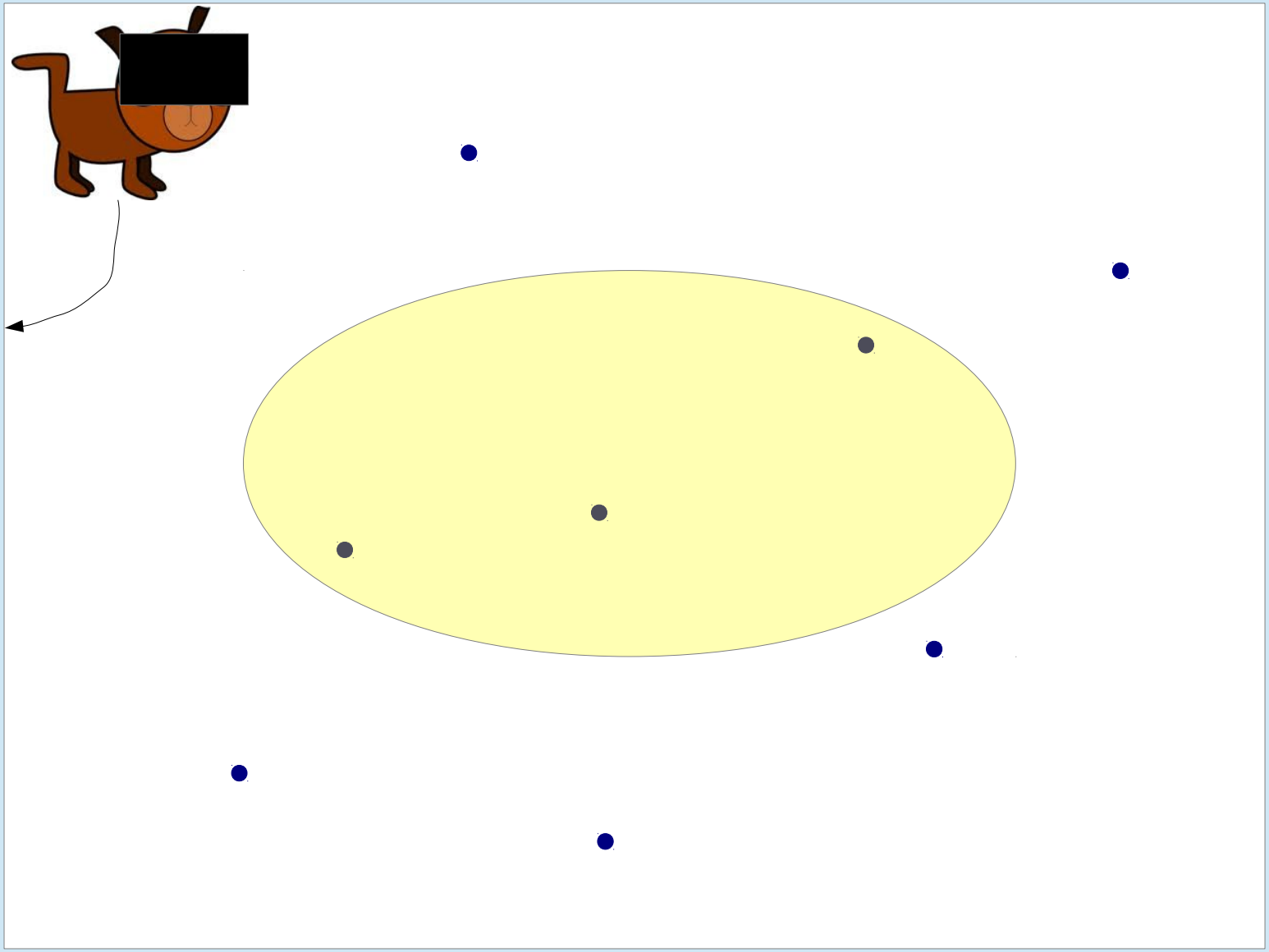

good solutions

Algorithm

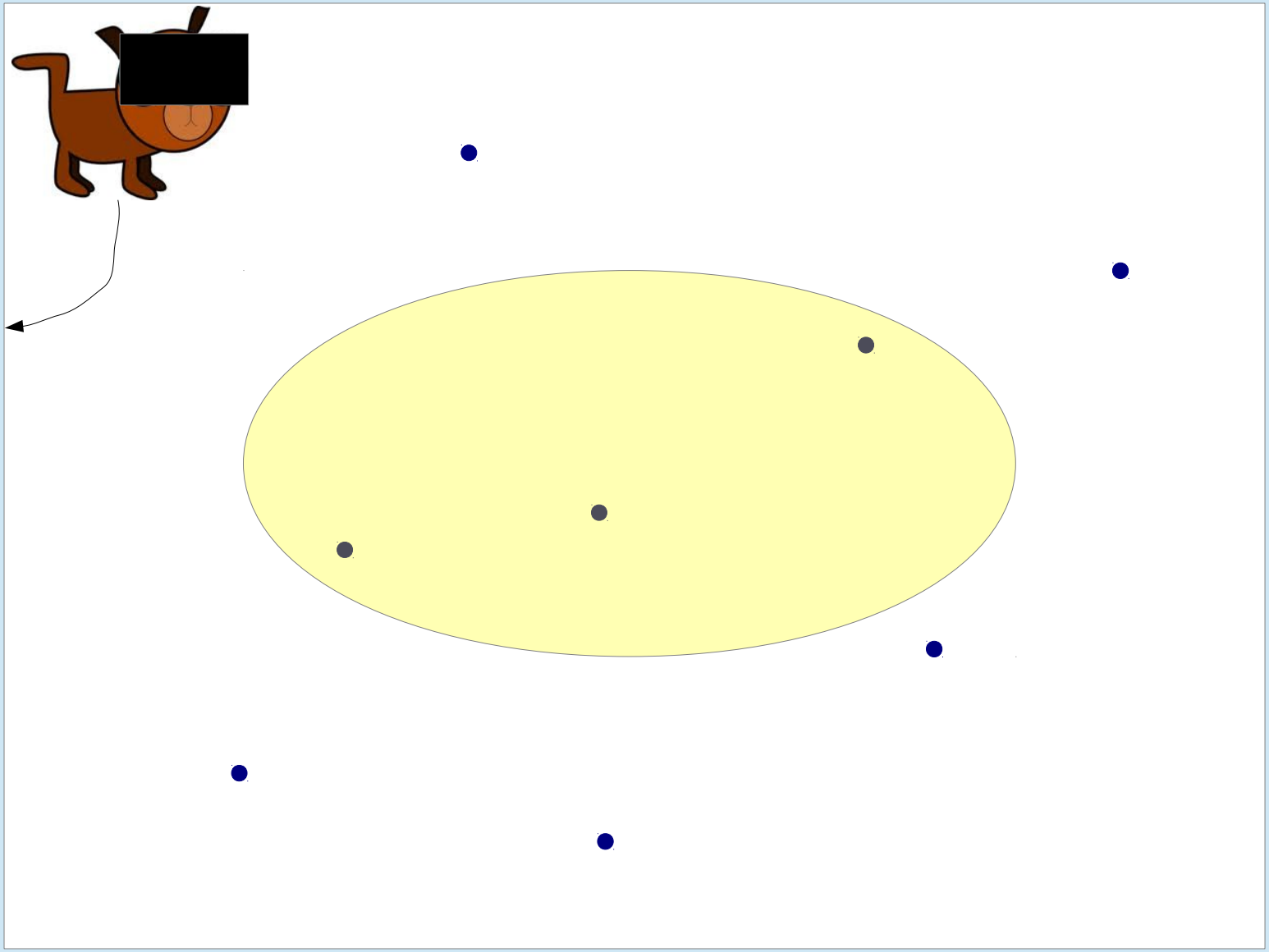


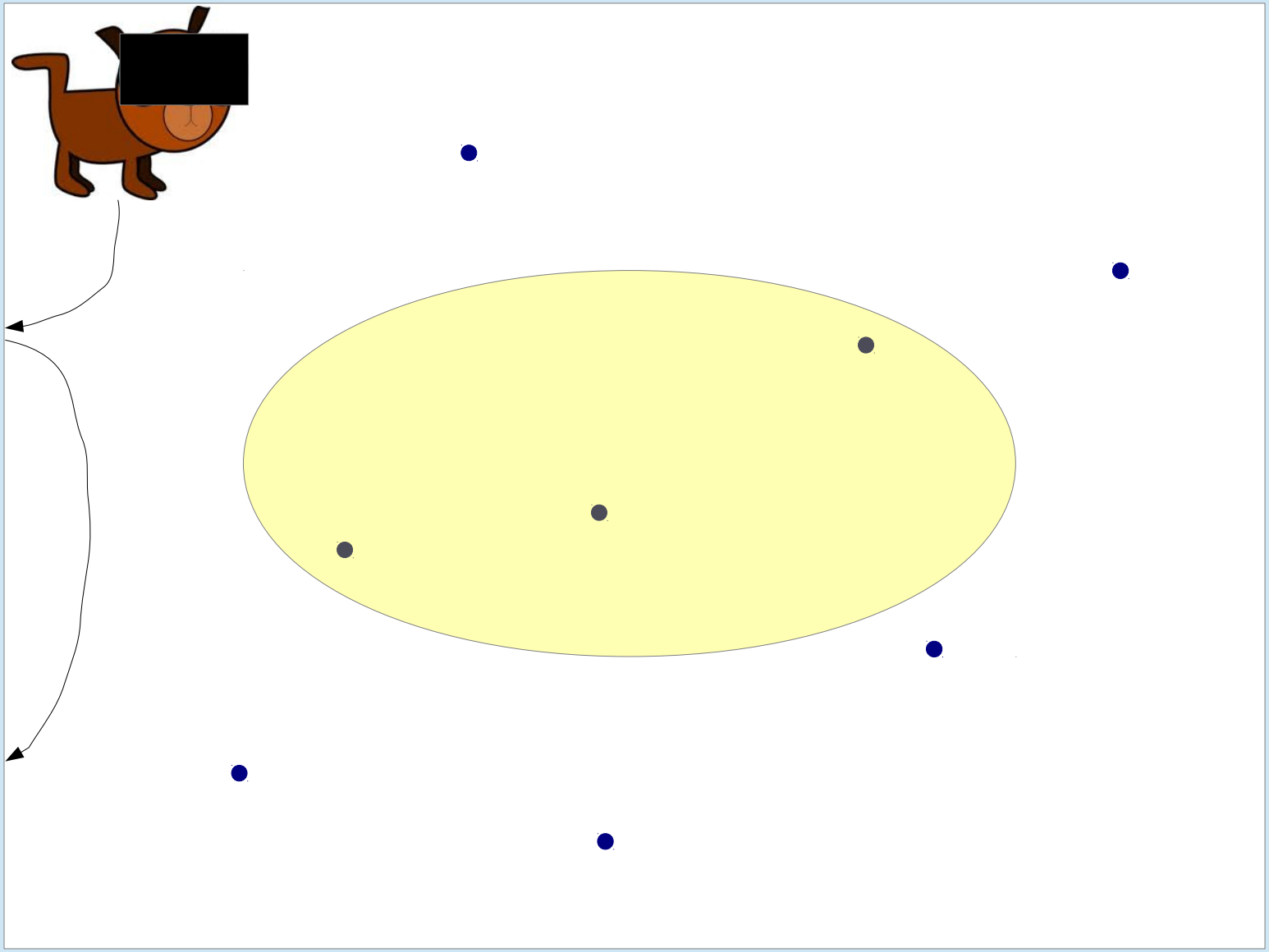
name: NR



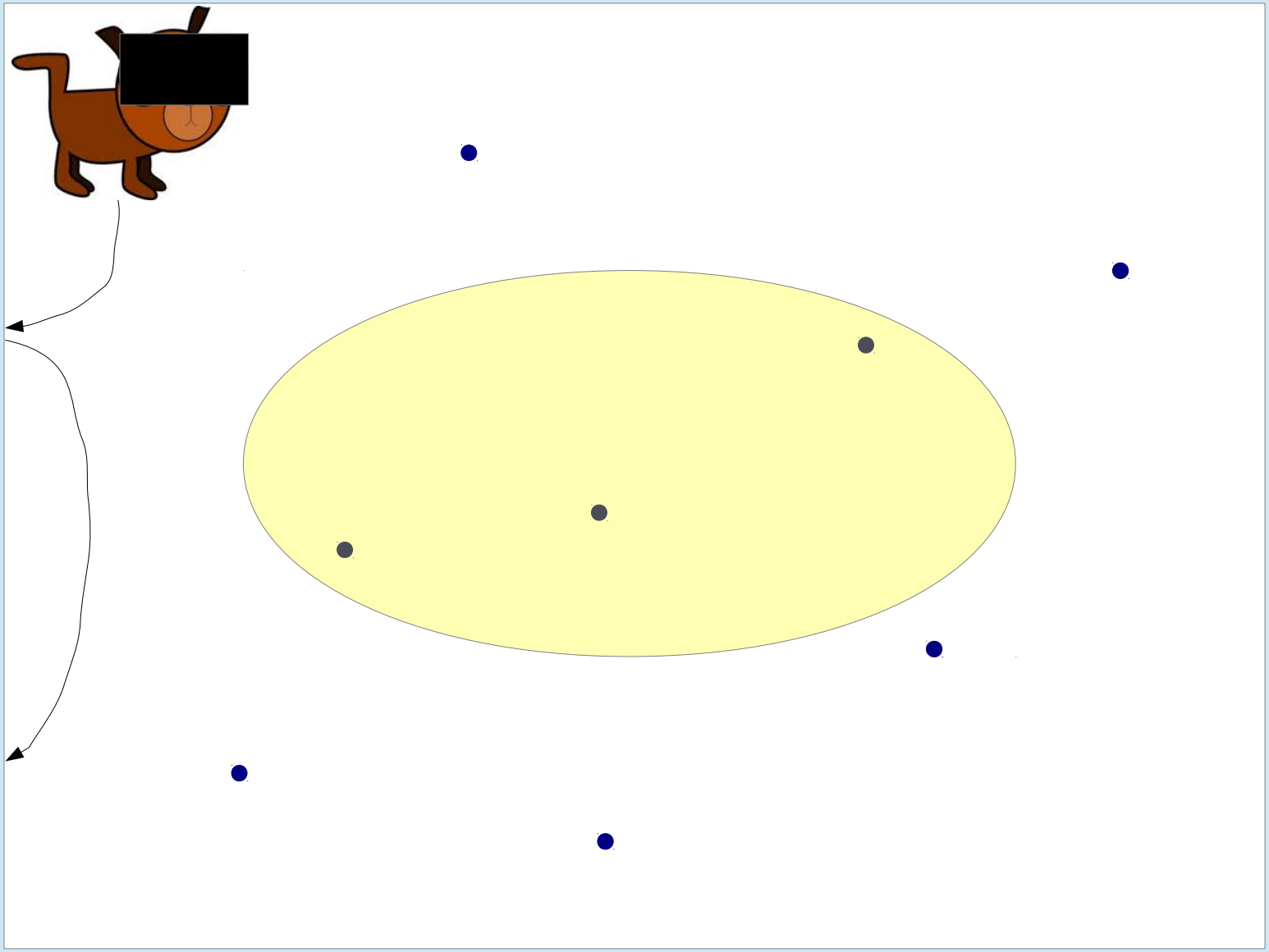


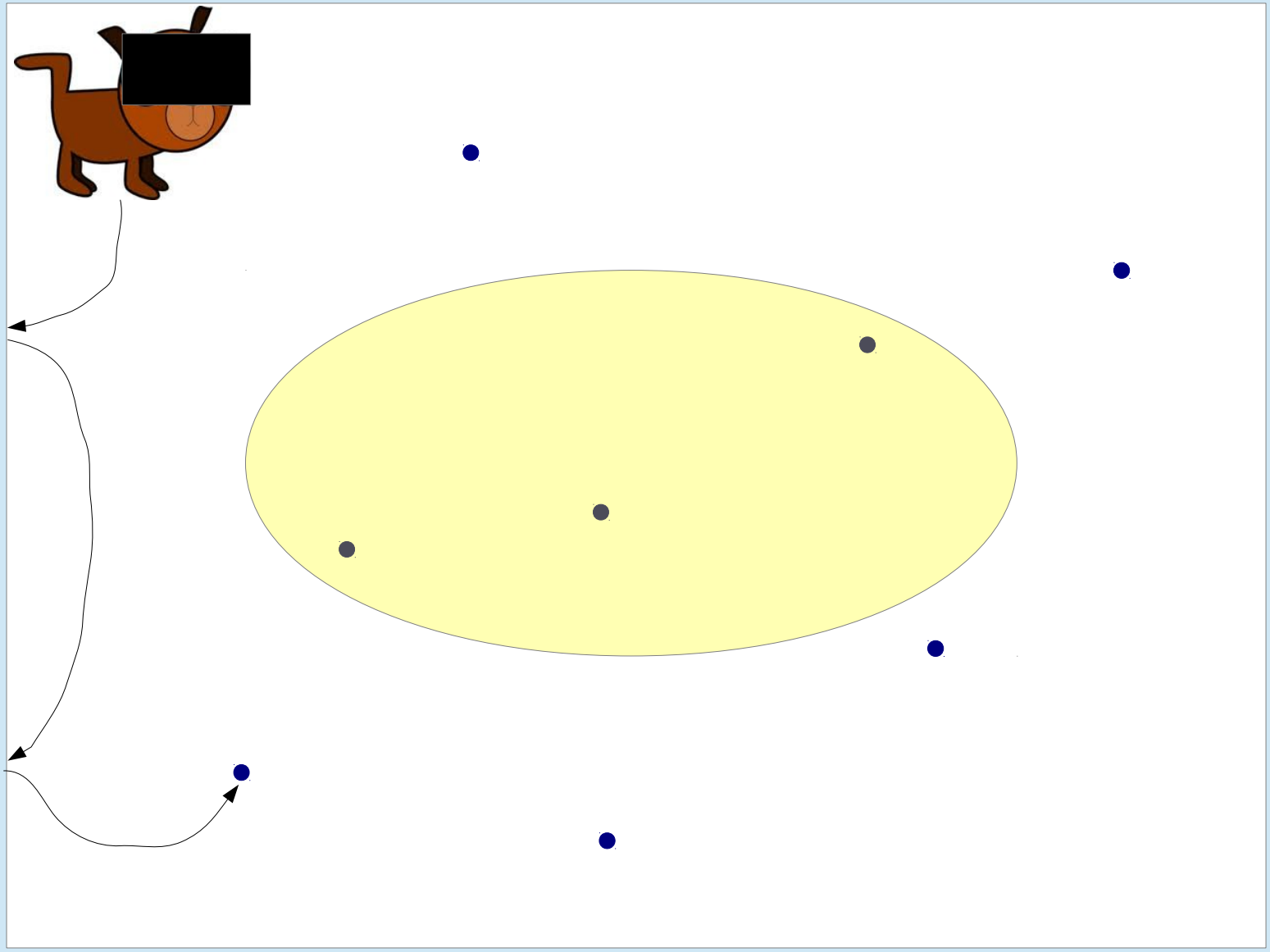

SWITCH!





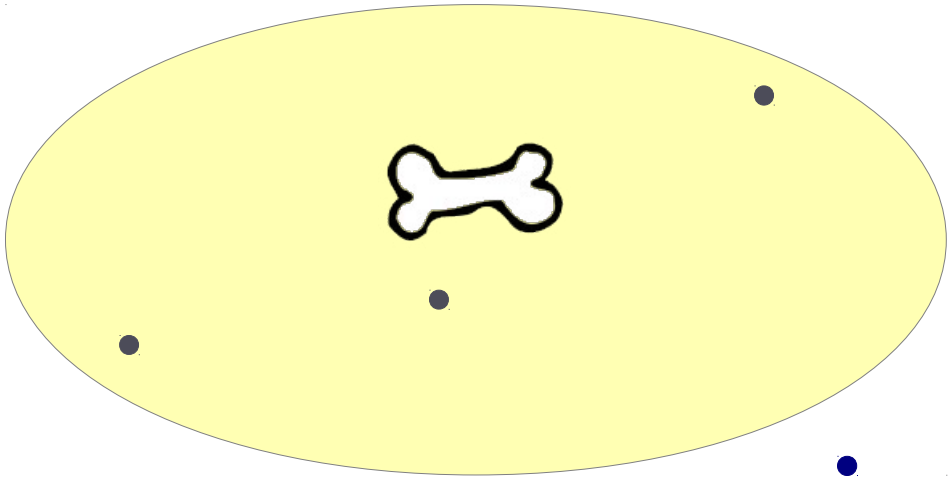

SWITCH!

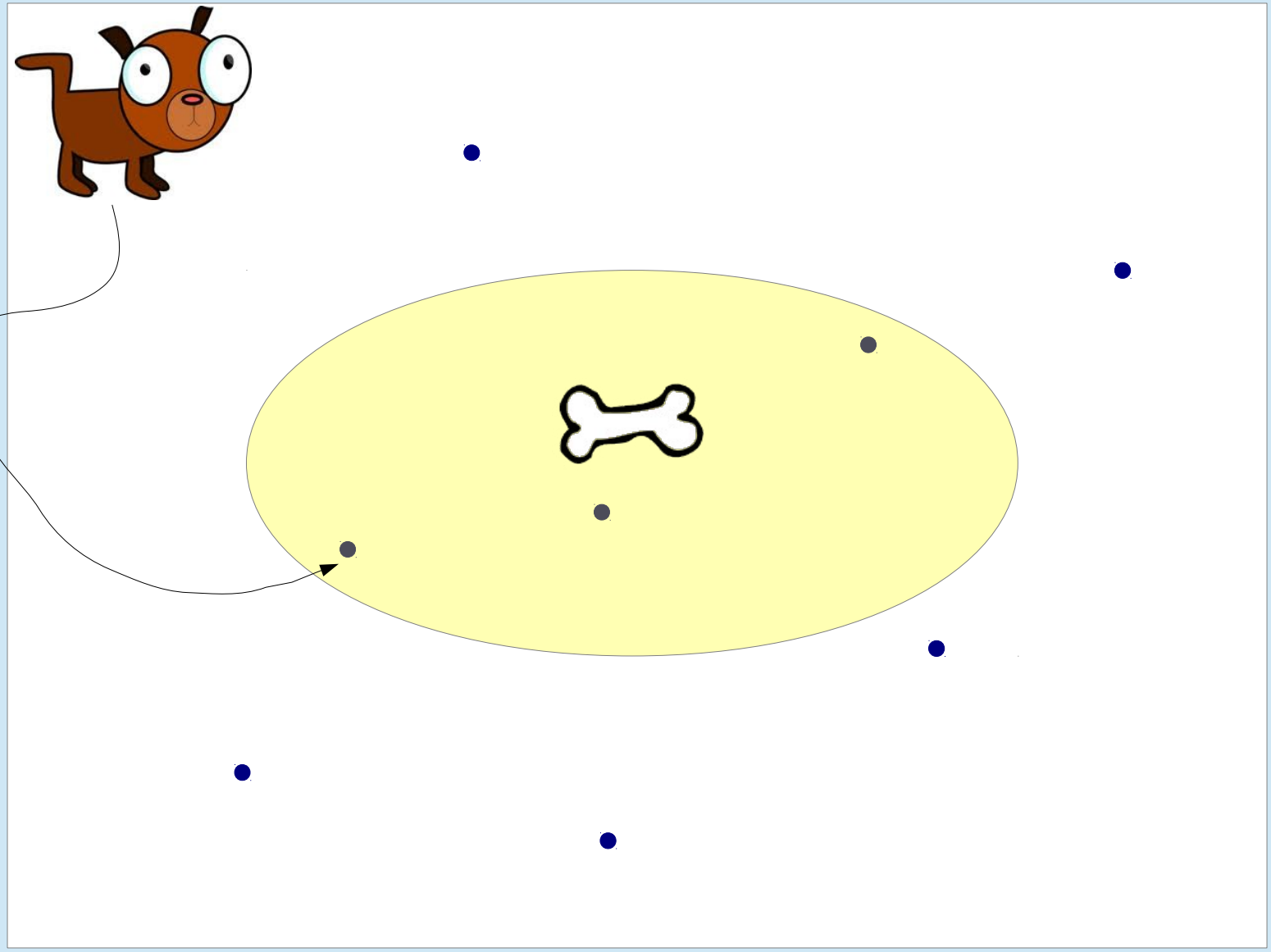




Algorithm

name: PBPF





vanishing regulation (VR)

vanishing regulation (VR)

formulate as optimization, encode preferences in objective

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \varphi(x) = \frac{\alpha}{2} \sum_{j \in \mathcal{R}} (v_j - v_j^t)^2 + \frac{\beta}{2} \sum_{j \in \mathcal{U}} (v_j - 1)^2 \\ \text{subject to} \quad & f(x) = 0 \end{aligned}$$

$\beta \ll \alpha$ (specific values unclear), x also includes $\{v_i\}_{i \in \mathcal{R}}$

vanishing regulation (VR)

formulate as optimization, encode preferences in objective

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \varphi(x) = \frac{\alpha}{2} \sum_{j \in \mathcal{R}} (v_j - v_j^t)^2 + \frac{\beta}{2} \sum_{j \in \mathcal{U}} (v_j - 1)^2 \\ \text{subject to} \quad & f(x) = 0 \end{aligned}$$

$\beta \ll \alpha$ (specific values unclear), x also includes $\{v_i\}_{i \in \mathcal{R}}$

apply penalty function method [23]

vanishing regulation (VR)

formulate as optimization, encode preferences in objective

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \varphi(x) = \frac{\alpha}{2} \sum_{j \in \mathcal{R}} (v_j - v_j^t)^2 + \frac{\beta}{2} \sum_{j \in \mathcal{U}} (v_j - 1)^2 \\ \text{subject to} \quad & f(x) = 0 \end{aligned}$$

$\beta \ll \alpha$ (specific values unclear), x also includes $\{v_i\}_{i \in \mathcal{R}}$

apply penalty function method [23]

approx. solve sequence

$$\underset{x}{\text{minimize}} \quad F_i(x) = \mu_i \varphi(x) + \frac{1}{2} \| f(x) \|_2^2$$

indexed by i, generate approx. solutions $\{x_i\}_{i \in \mathbb{N}}$

solving subproblems: apply LSNR to $\nabla F_i(x) = 0$

solving subproblems: apply LSNR to $\nabla F_i(x) = 0$

LSNR iteration needs to solve

$$\Big(\mu_i G_k + J_k^T J_k + \sum_j f_j(x_k)\nabla^2 f_j(x_k)\Big)p_k = -\mu_k g_k - J_k^T f_k,$$

G_k and g_k are Hessian and gradient of φ at x_k

solving subproblems: apply LSNR to $\nabla F_i(x) = 0$

LSNR iteration needs to solve

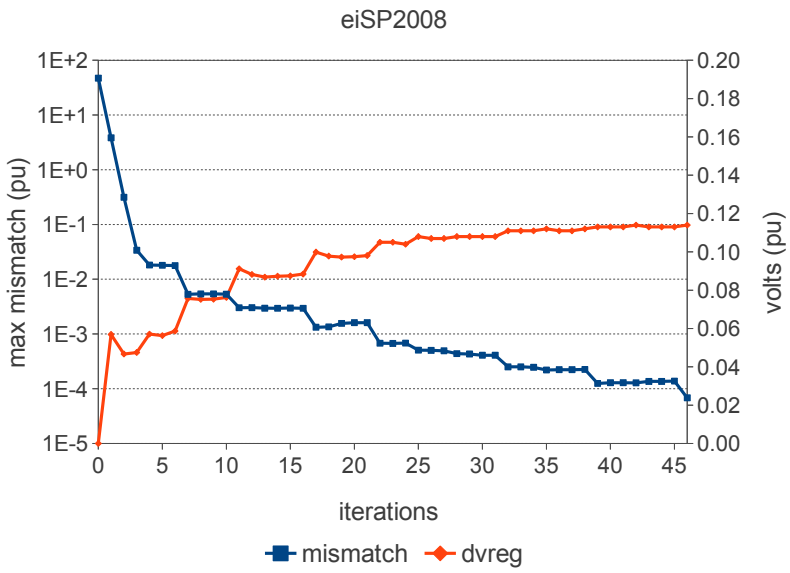
$$\Big(\mu_i G_k + J_k^T J_k + \sum_j f_j(x_k)\nabla^2 f_j(x_k)\Big)p_k = -\mu_k g_k - J_k^T f_k,$$

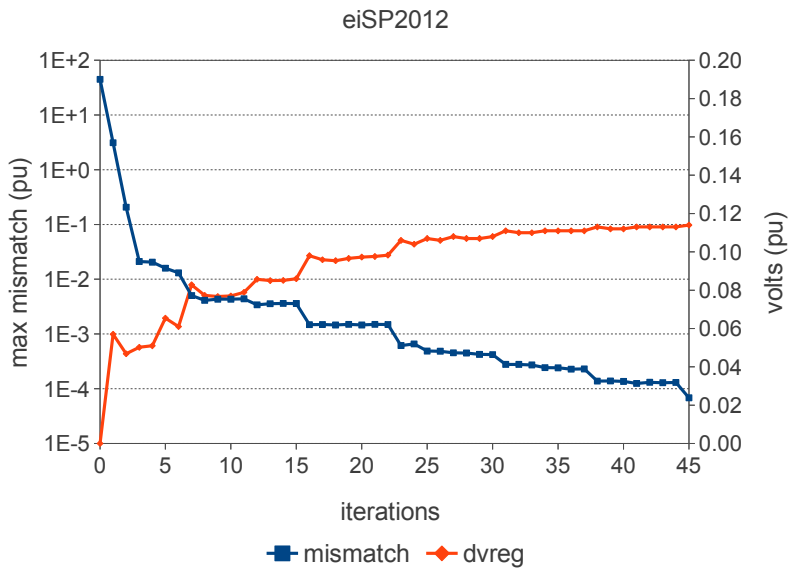
G_k and g_k are Hessian and gradient of φ at x_k

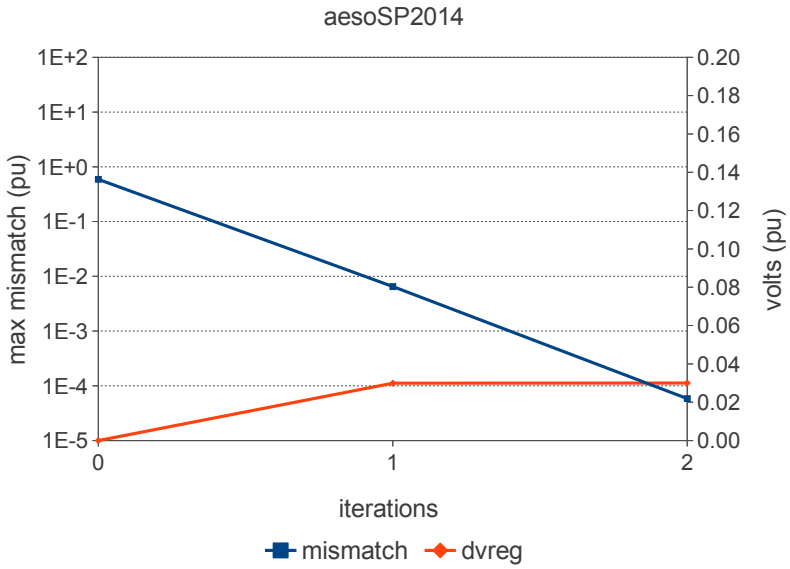
VR ignores $\sum_j f_j(x_k)\nabla^2 f_j(x_k)$ to get descent directions easily, but then

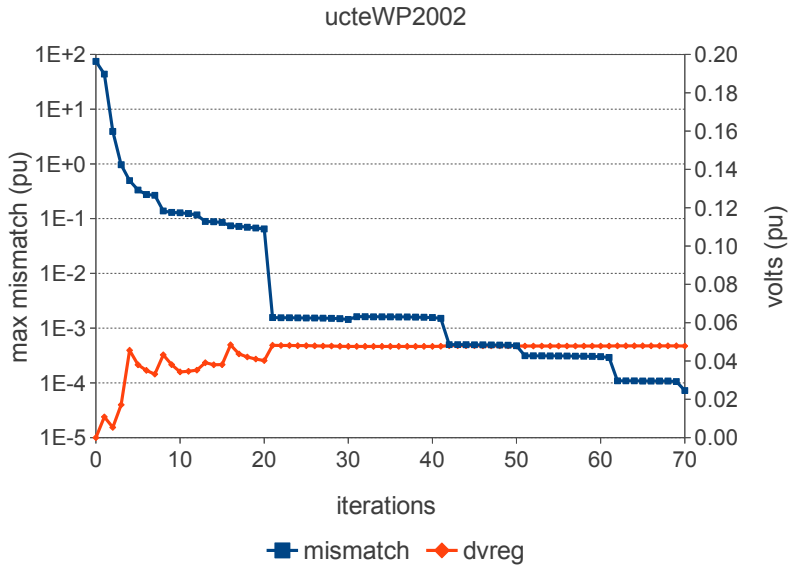
- descent directions get poorer as $\mu_i \downarrow 0$ ($J_k^T J_k$ singular)
- affects robustness of method

mismatch vs regulation trade-off









penalty-based power flow (PBPF)

penalty-based power flow (PBPF)

extend VR:

- consider physical limits $Q_i^{\min} \leq Q_i^g \leq Q_i^{\max}$, $i \in \mathcal{R}$

penalty-based power flow (PBPF)

extend VR:

- consider physical limits $Q_i^{\min} \leq Q_i^g \leq Q_i^{\max}$, $i \in \mathcal{R}$

improve VR:

- use better optimization algorithm
- keep second order terms in Newton system
- use better penalties to encode preferences

formulate as optimization, encode preferences & physical limits in objective

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \varphi(x) = \varphi^u(x) + \varphi^q(x) + \varphi^r(x) \\ \text{subject to} \quad & f(x) = 0 \end{aligned}$$

x also includes $\{Q_i\}_{i \in \mathcal{R}}$

f is now mismatches at all buses except slack

formulate as optimization, encode preferences & physical limits in objective

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \varphi(x) = \varphi^u(x) + \varphi^q(x) + \varphi^r(x) \\ \text{subject to} \quad & f(x) = 0 \end{aligned}$$

x also includes $\{Q_i\}_{i \in \mathcal{R}}$

f is now mismatches at all buses except slack

penalties:

$$\varphi^u(x) = \sum_{j \in \mathcal{U}} \varphi_j^u(v_j)$$

$$\varphi^q(x) = \sum_{j \in \mathcal{R}} \varphi_j^q(Q_j^g)$$

$$\varphi^r(x) = \sum_{j \in \mathcal{R}} \varphi_j^r(v_j)$$

for unregulated magnitudes (quartic)

$$\varphi_j^u(z) = \left(\frac{2z - v_j^{\max} - v_j^{\min}}{v_j^{\max} - v_j^{\min}} \right)^4$$

for unregulated magnitudes (quartic)

$$\varphi_j^u(z) = \left(\frac{2z - v_j^{\max} - v_j^{\min}}{v_j^{\max} - v_j^{\min}} \right)^4$$

for reactive powers of regulating gens (sextic)

$$\varphi_j^q(z) = \left(\frac{2z - Q_j^{\max} - Q_j^{\min}}{Q_j^{\max} - Q_j^{\min}} \right)^6$$

for unregulated magnitudes (quartic)

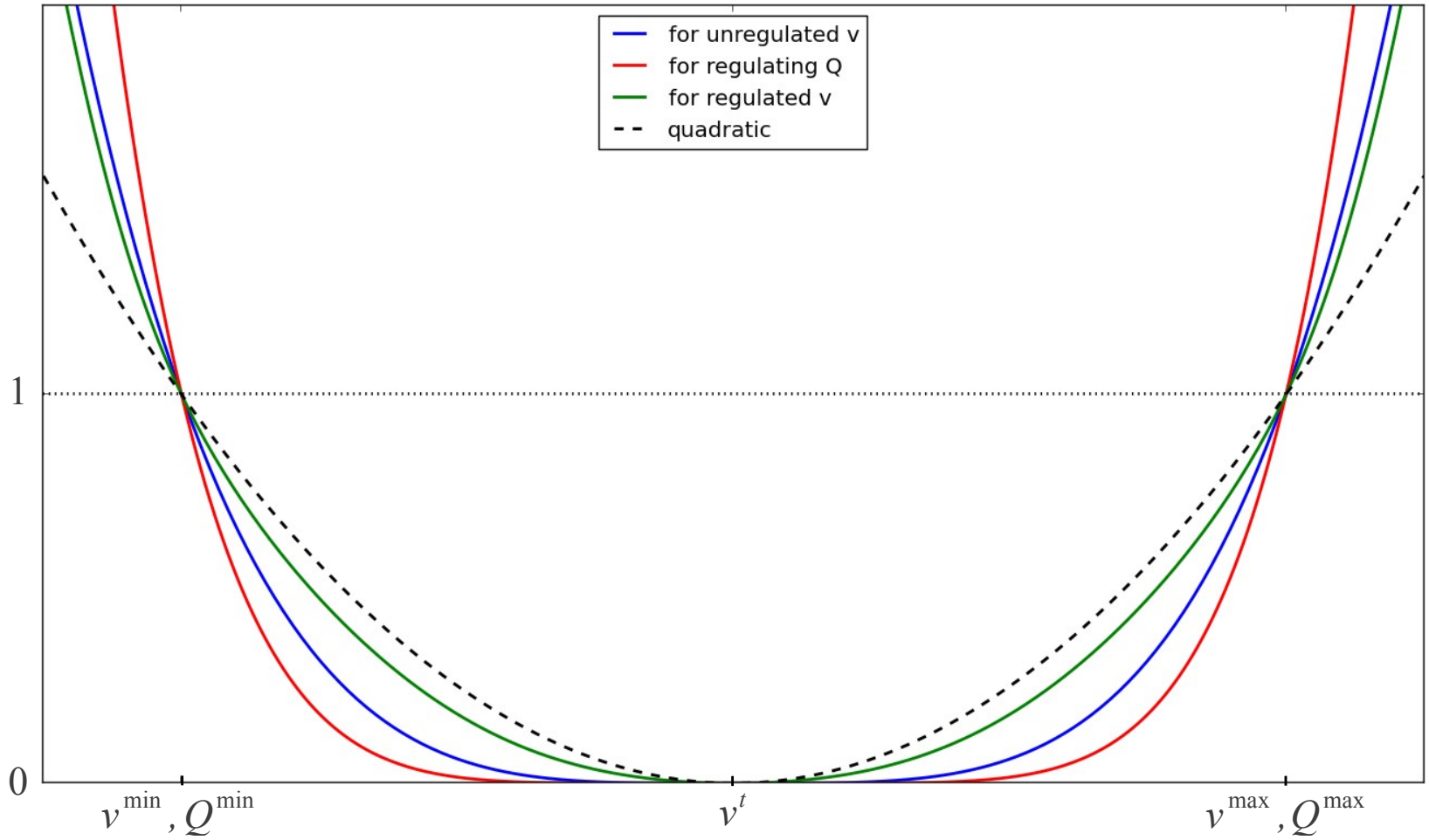
$$\varphi_j^u(z) = \left(\frac{2z - v_j^{\max} - v_j^{\min}}{v_j^{\max} - v_j^{\min}} \right)^4$$

for reactive powers of regulating gens (sextic)

$$\varphi_j^q(z) = \left(\frac{2z - Q_j^{\max} - Q_j^{\min}}{Q_j^{\max} - Q_j^{\min}} \right)^6$$

for regulated magnitudes (softmax between quartic & quadratic)

$$\varphi_j^r(z) = \frac{1}{2} \log \left(e^{2\varphi_j^u(z)} + e^{2\varphi_j^2(z)} \right) + b_j,$$


for unregulated v
for regulating Q
for regulated v
quadratic
1
0
$v^{\min}, Q^{\min}$
v^t
$v^{\max}, Q^{\max}$

apply augmented Lagrangian method [8][10][22][23]

apply augmented Lagrangian method [8][10][22][23]

approx. solve sequence

$$\underset{x}{\text{minimize}} \quad L_{\mu_i}(x, \lambda_i) = \mu_i \varphi(x) - \mu_i \lambda_i^T f(x) + \frac{1}{2} \|f(x)\|_2^2,$$

indexed by i, generate approx. solutions $\{x_i\}_{i \in \mathbb{N}}$

μ_i positive scalars, λ_i Lagrange multiplier estimates

apply augmented Lagrangian method [8][10][22][23]

approx. solve sequence

$$\underset{x}{\text{minimize}} \quad L_{\mu_i}(x, \lambda_i) = \mu_i \varphi(x) - \mu_i \lambda_i^T f(x) + \frac{1}{2} \|f(x)\|_2^2,$$

indexed by i, generate approx. solutions $\{x_i\}_{i \in \mathbb{N}}$

μ_i positive scalars, λ_i Lagrange multiplier estimates

solving subproblems: apply truncated LSNR procedure [33][37]

apply augmented Lagrangian method [8][10][22][23]

approx. solve sequence

$$\underset{x}{\text{minimize}} \quad L_{\mu_i}(x, \lambda_i) = \mu_i \varphi(x) - \mu_i \lambda_i^T f(x) + \frac{1}{2} \|f(x)\|_2^2,$$

indexed by i, generate approx. solutions $\{x_i\}_{i \in \mathbb{N}}$

μ_i positive scalars, λ_i Lagrange multiplier estimates

solving subproblems: apply truncated LSNR procedure [33][37]

truncated LSNR iteration solves $H_k p_k = -d_k$ only approx.

- H_k is $\nabla_x^2 L_{\mu_i}(x_k, \lambda_i)$, d_k is $\nabla_x L_{\mu_i}(x_k, \lambda_i)$
- x_k is current iterate

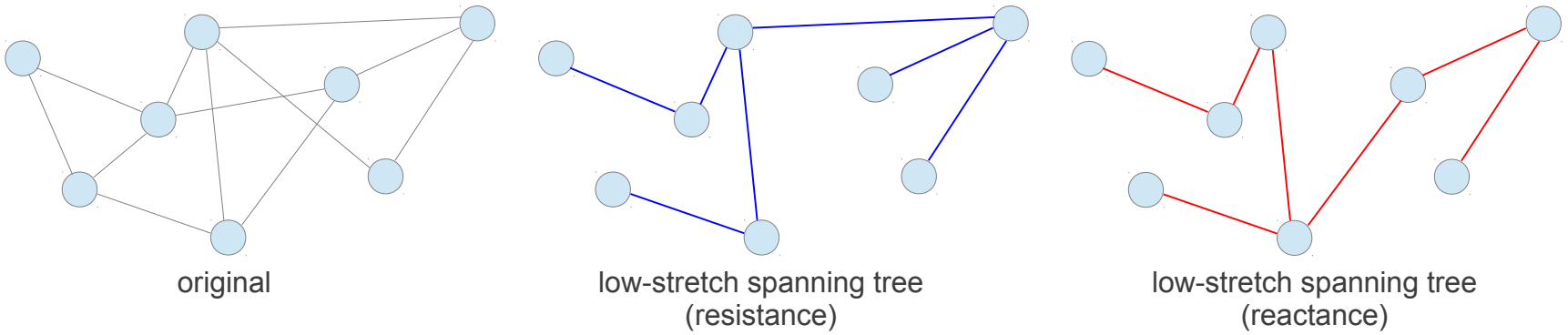
we use | modified Conjugate Gradient | [12]

- exits with $p_k = -\Phi_k d_k$, $\Phi_k \succeq 0$, sufficient descent direction for $L_{\mu_i}(\cdot, \lambda_i)$

we use modified Conjugate Gradient [12]

- exits with $p_k = -\Phi_k d_k$, $\Phi_k \succeq 0$, sufficient descent direction for $L_{\mu_i}(\cdot, \lambda_i)$

preconditioner:



merge low-stretch spanning trees [4][21], find $\tilde{H}_k$ based on resulting tree

- captures important info of H_k and sparser

factorize $\tilde{H}_k = LDL^T$, modify D if necessary (positive definite)

3.4. real networks

3.4. real networks

most common aspect of methods from literature:

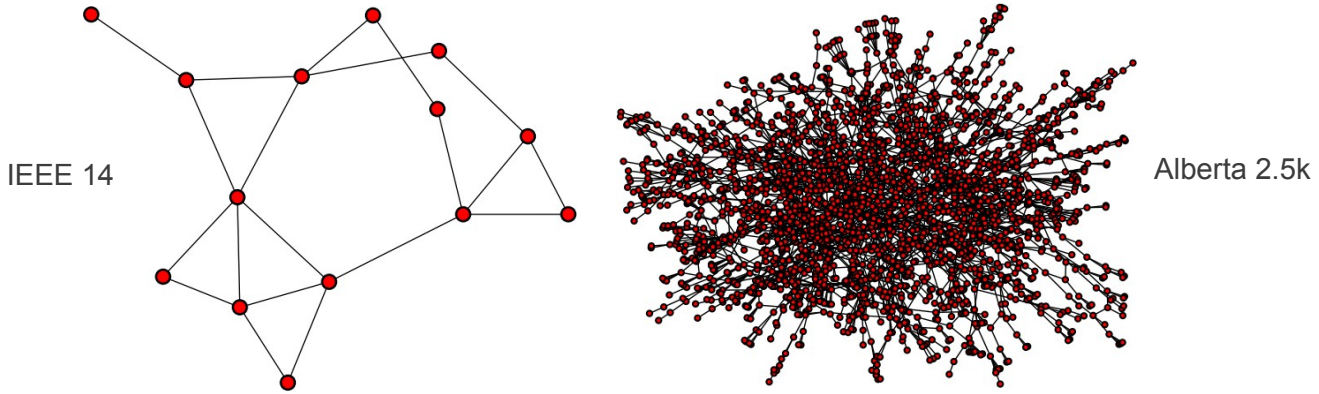
- tested on small toy problems (IEEE networks [1])

3.4. real networks

most common aspect of methods from literature:

- tested on small toy problems (IEEE networks [1])

in reality: networks can be large (some $> 45k$ buses)



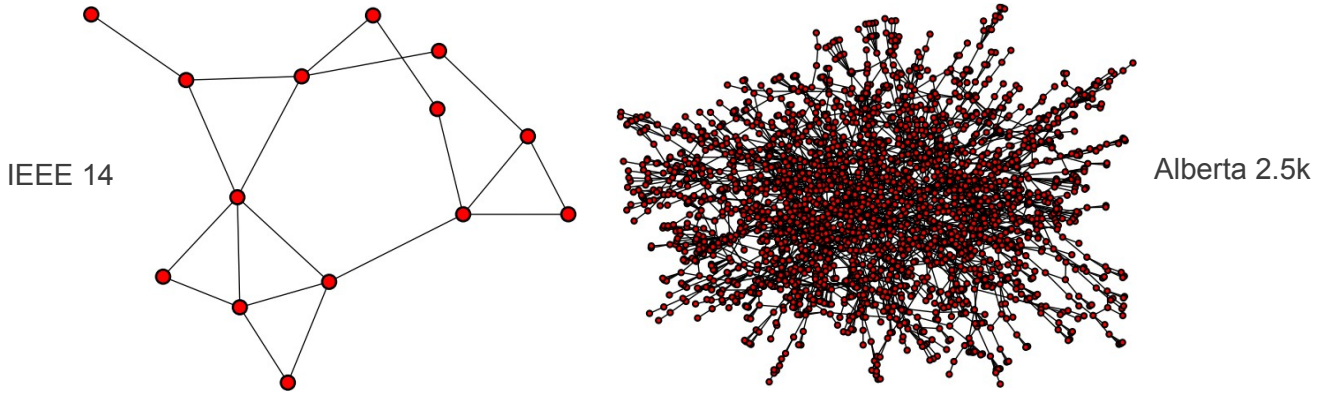
performance on toy problems gives little info

3.4. real networks

most common aspect of methods from literature:

- tested on small toy problems (IEEE networks [1])

in reality: networks can be large (some $> 45k$ buses)



performance on toy problems gives little info

our data: real power networks

4. implementation

4. implementation

algorithms

- Python [2] (fast prototyping)
- objects and basic numerical routines with NumPy [3] and SciPy [30]
- efficient sparse LU with UMFPACK [16] [17] [18] [19]
- analysis of topology and modifications with NetworkX [25]

4. implementation

algorithms

- Python [2] (fast prototyping)
- objects and basic numerical routines with NumPy [3] and SciPy [30]
- efficient sparse LU with UMFPACK [16] [17] [18] [19]
- analysis of topology and modifications with NetworkX [25]

parsers

- also in Python
- handle IEEE common data format [24] and PSS®E raw format version 32

5. experiments

5. experiments

power flow test cases

name	buses	branches
ieee118	118	186
ieee300	300	411
ucteSL2002	1254	1944
ucteWL2002	1254	1944
ucteWP2002	1254	1944
aesoSL2014	2495	2823
aesoSP2014	2495	2823
aesoWP2014	2529	2869
eiSP2008	46197	60177
eiSP2012	43792	57483

method comparison (warm start)

name	LSNR	PSSE	PSSE Q-lim	VR	PBPF
ieee118					
ieee300					
ucteSL2002					
ucteWL2002					
ucteWP2002					
aesoSL2014					
aesoSP2014					
aesoWP2014					
eiSP2008					
eiSP2012					

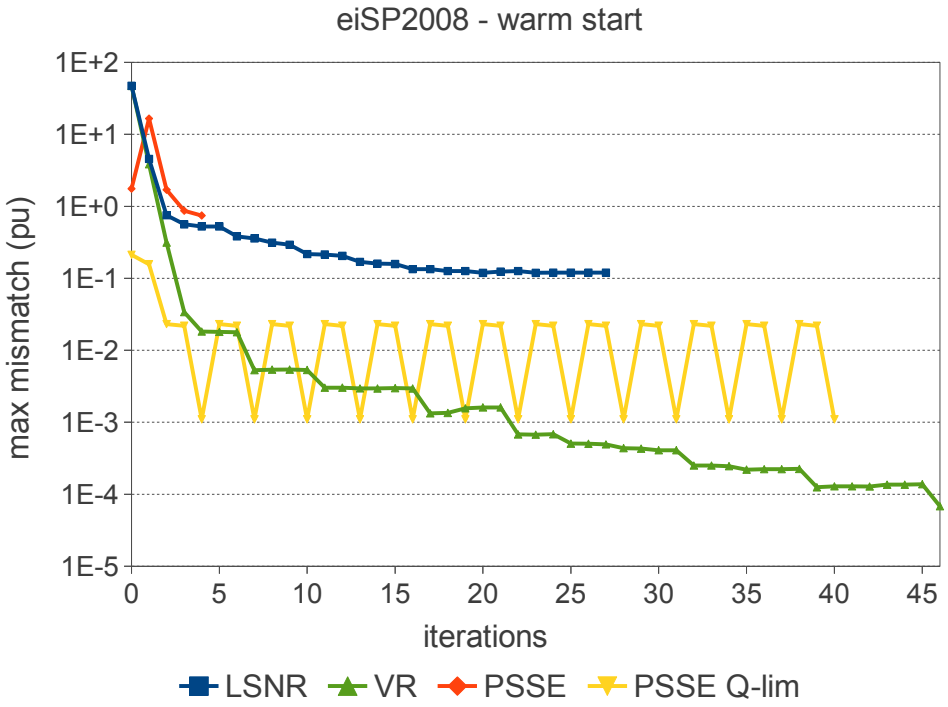
	converged
	failed
	unavailable

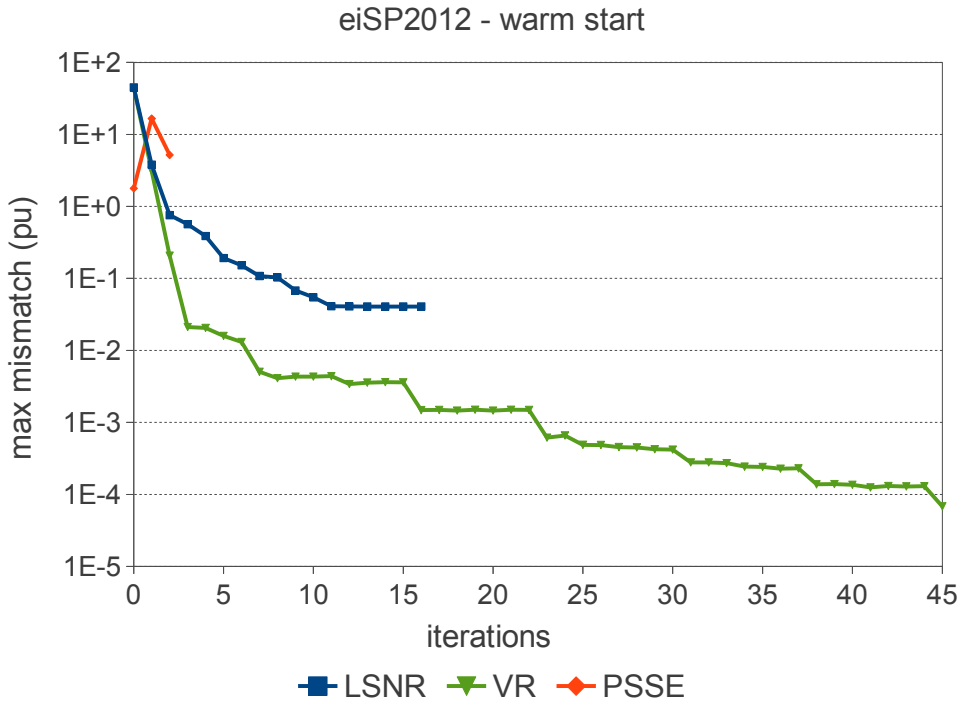
method comparison (flat start)

name	LSNR	PSSE	FNH	PSSE Q-lim	VR	PBPF
ieee118						
ieee300						
ucteSL2002						
ucteWL2002						
ucteWP2002						
aesoSL2014						
aesoSP2014						
aesoWP2014						
eiSP2008						
eiSP2012						

	converged
	failed
	unavailable

method comparison (progress on large cases)





method comparison (perturbing x_0)

aesoSL2014			
seed	NR	LSNR	VR
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			

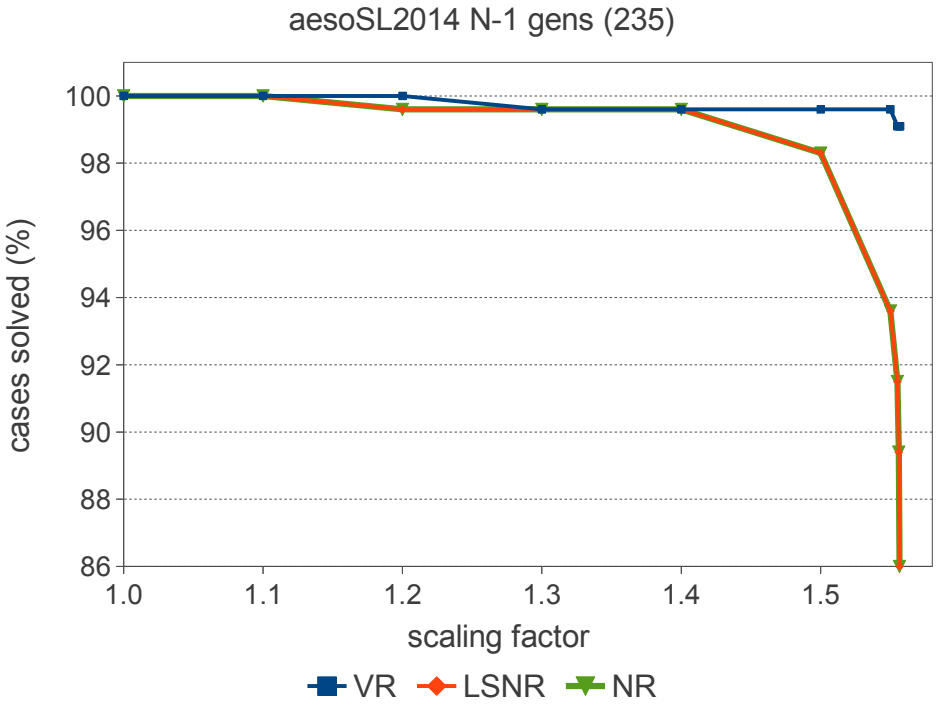
initial angle + $\mathcal{N}(0, 0.3^2)$ (degrees)

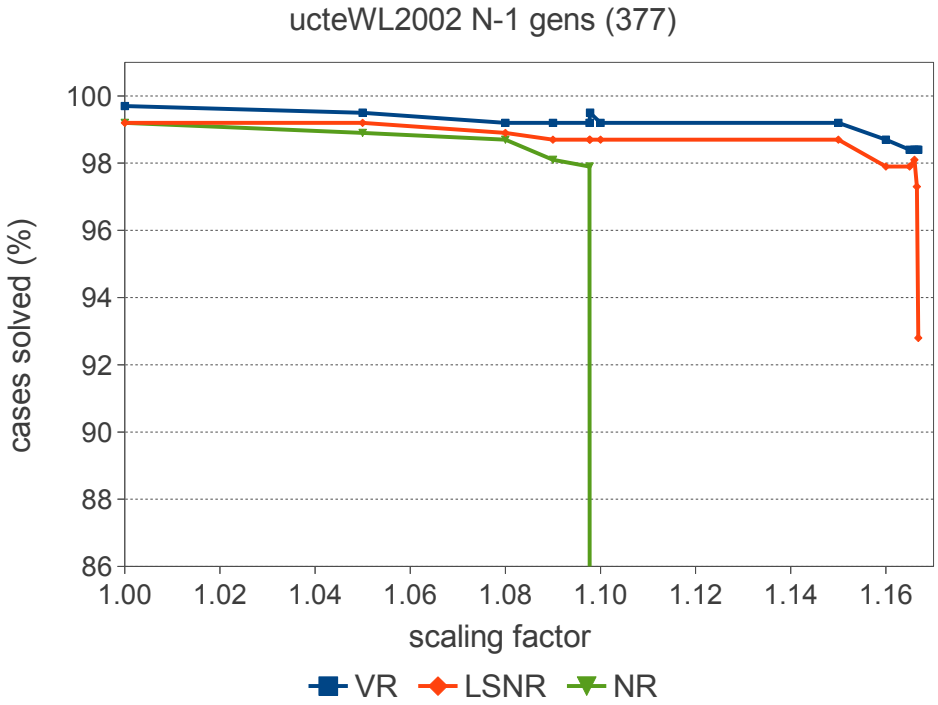
method comparison (perturbing x_0)

ucteWL2002			
seed	NR	LSNR	VR
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			

initial angle $+ \mathcal{N}(0, 0.5^2)$ (degrees)

contingency analysis (scaling net injections)





6. conclusions

line search:

- simple to implement
- not expensive
  - only a few function evaluations
    - easy to parallelize
- big value

flat network homotopy:

- phase shifters are important but not enough
- need more transformations to simplify problem
- try to make DC power flow assumptions [39] hold:
  - near flat θ profile
  - near flat v profile
  - small r and high x/r ratio

flat network homotopy:

- phase shifters are important but not enough
- need more transformations to simplify problem
- try to make DC power flow assumptions [39] hold:
  - near flat θ profile
  - near flat v profile
  - small r and high x/r ratio

to do

- understand θ & v profile as net injections vary
- understand θ & v profile as r becomes small and x/r large
- use homotopy with net injections, r, x and phase shifters

formulation and bus type switching

- moving away from square formulation is promising
- more robust, no more switching, preferred solutions
- VR has limitations, PBPF should solve them

formulation and bus type switching

- moving away from square formulation is promising
- more robust, no more switching, preferred solutions
- VR has limitations, PBPF should solve them

to do

- finish developing theory for PBPF
- start implementing PBPF
- parallelize function evaluations
- combine with FNH

References

[1] Power systems test case archive. `http://www.ee.washington.edu/research/pstca/`. Accessed: 08/05/2012.

[2] Python programming language. `http://www.python.org`. Accessed: 10/8/2012.

[3] Scientific computing with Python. `http://numpy.scipy.org/`. Accessed: 10/8/2012.

[4] Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *Proceedings of the 49th Annual Symposium on Foundations of Computer Science*, pages 781–790, 2008.

[5] Enrique Acha. *FACTS: Modelling and simulation in power networks*. Wiley, 2004.

[6] Göran Andersson. Modelling and analysis of electric power systems. *ETH Zurich*, september 2008.

[7] George A. Baker and Peter Graves-Morris. *Padé Approximants*. Cambridge University Press, 1996.

[8] D.P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Computer Science and Applied Mathematics. Academic Press, 1982.

[9] P.R. Bijwe and S.M. Kelapure. Nondivergent fast power flow methods. *Power Systems, IEEE Transactions on*, 18(2):633 – 638, may 2003.

[10] E. G. Birgin and J. M. Martínez. Practical augmented lagrangian methods, 2007.

[11] L.M.C. Braz, C.A. Castro, and C.A.F. Murati. A critical evaluation of step size optimization based load flow methods. *Power Systems, IEEE Transactions on*, 15(1):202 –207, feb 2000.

[12] Laurent Chauvier, Antonio Fuduli, and Jean Charles Gilbert. A truncated sqp algorithm for solving nonconvex equality constrained optimization problems, 2001.

[13] Ying Chen and Chen Shen. A Jacobian-free Newton-GMRES(m) method with adaptive preconditioner and its application for power flow calculations. *Power Systems, IEEE Transactions on*, 21(3):1096 –1103, august 2006.

[14] H. Dag and A. Semlyen. A new preconditioned conjugate gradient power flow. *Power Systems, IEEE Transactions on*, 18(4):1248 – 1255, november 2003.

[15] H. Dag and and E.F. Yetkin. A spectral divide and conquer method based preconditioner design for power flow analysis. In *Power System Technology (POWERCON), 2010 International Conference on*, pages 1 – 6, october 2010.

[16] Timothy A. Davis. Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):196–199, june 2004.

[17] Timothy A. Davis. A column pre-ordering strategy for the unsymmetric-

pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):165–195, june 2004.

[18] Timothy A. Davis and Iain S. Duff. An unsymmetric-pattern multifrontal method for sparse lu factorization. *SIAM J. Matrix Anal. Appl.*, 18(1):140–158, january 1997.

[19] Timothy A. Davis and Iain S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. Math. Softw.*, 25(1):1–20, march 1999.

[20] F. De Leon and A. Sernlyen. Iterative solvers in the Newton power flow problem: preconditioners, inexact solutions and partial Jacobian updates. *Generation, Transmission and Distribution, IEEE Proceedings on*, 149(4):479 – 484, july 2002.

[21] Michael Elkin, Daniel A. Spielman, Yuval Emek, and Shang hua Teng. Lower-stretch spanning trees. In *STOC 05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 494–503. ACM Press, 2005.

[22] R. Fletcher. *Practical Methods of Optimization*. A Wiley-Interscience Publication. Wiley, 2000.

[23] P.E. Gill, W. Murray, and M.H. Wright. *Practical optimization*. Academic Press, 1981.

[24] W. Group. Common format for exchange of solved load flow data. *Power Apparatus and Systems, IEEE Transactions on*, PAS-92(6):1916 –1925, nov. 1973.

[25] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[26] Reijer Idema, Domenico Lahaye, Kees Vuik, and Lou van der Sluis. Fast Newton load flow. In *Transmission and Distribution Conference and Exposition, 2010 IEEE PES*, pages 1 –7, april 2010.

[27] S. Iwamoto and Y. Tamura. A load flow calculation method for ill-conditioned power systems. *Power Apparatus and Systems, IEEE Transactions on*, PAS-100(4):1736 –1743, april 1981.

[28] R.A. Jabr. Radial distribution load flow using conic programming. *Power Systems, IEEE Transactions on*, 21(3):1458 – 1459, august 2006.

[29] R.A. Jabr. A conic quadratic format for the load flow equations of meshed networks. *Power Systems, IEEE Transactions on*, 22(4):2285 – 2286, november 2007.

[30] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[31] R. Kalaba and L. Tesfatsion. Solving nonlinear equations by adaptive homotopy continuation. *Applied Mathematics and Computation*, 41(2):99 – 115, 1991.

[32] Amirhassan Kamiabad. Implementing a preconditioned iterative linear solver using massively parallel graphics processing units. Master's thesis,

University of Toronto, 2003. https://tspace.library.utoronto.ca/handle/1807/27321.

[33] Stefano Lucidi and Massimo Roma. Numerical experiences with new truncated newton methods in large scale unconstrained optimization. In Almerico Murli and Gerardo Toraldo, editors, *Computational Issues in High Performance Software for Nonlinear Optimization*, pages 71–87. Springer US, 1997.

[34] F. Milano. Continuous Newton's method for power flow analysis. *Power Systems, IEEE Transactions on*, 24(1):50 – 57, february 2009.

[35] H. Mori and F. Iizuka. An ILU(p)-preconditoner Bi-CGStab method for power flow calculation. In *Power Tech, 2007 IEEE Lausanne*, pages 1474 – 1479, july 2007.

[36] Walter Murray. *Newton-Type Methods*. John Wiley & Sons, Inc., 2010.

[37] Stephen G. Nash. *Truncated-Newton Methods*. PhD thesis, Stanford University, 1982.

[38] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, august 2000.

[39] K. Purchala, L. Meeus, D. Van Dommelen, and R. Belmans. Usefulness of dc power flow for active power flow analysis. In *Power Engineering Society General Meeting, 2005. IEEE*, pages 454 – 459 Vol. 1, june 2005.

[40] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2 edition, april 2003.

[41] J. Scudder and F.L. Alvarado. *Step Size Optimization in a Polar Newton Power Flow*. Wisconsin. University - Madison. Department of Electrical and Computer Engineering. [Papers]. University of Wisconsin, Engineering Experiment Station, 1981.

[42] J.E. Tate and T.J. Overbye. A comparison of the optimal multiplier in polar and rectangular coordinates. *Power Systems, IEEE Transactions on*, 20(4):1667 – 1674, november 2005.

[43] Antonio Trias. The holomorphic embedding load flow method. *IEEE PES general meeting*, july 2012. `http://www.gridquant.com/technology/`.

[44] X.F. Wang, Y. Song, and M. Irving. *Modern Power Systems Analysis*. Power Electronics and Power Systems. Springer, 2008.

[45] Yi-Shan Zhang and Hsiao-Dong Chiang. Fast Newton-FGMRES solver for large-scale power flow study. *Power Systems, IEEE Transactions on*, 25(2):769 – 776, may 2010.

[46] Jinquan Zhao, Hsiao-Dong Chiang, Ping Ju, and Hua Li. On PV-PQ bus type switching logic in power flow computation. In *Power Systems Computation Conference (PSCC)*, 2008.

[47] J. Zhu. *Optimization of Power System Operation*. IEEE Press Series on Power Engineering. Wiley, 2009.