Numerical linear algebra for large spatial problems

Daniel Simpson
Institutt for matematiske fag
2 December, 2010
Outline

Introduction

Newton’s Method

Solving linear systems with Krylov subspaces

Conclusion

Extras
A common situation

Observed field \( y = g(x) + \text{noise} \)

Latent field \( x \)

Parameters \( \theta \)

Question: CAN WE INFER \( x \) FROM \( y \) WHEN THE SIZE OF \( x \) AND \( y \) IS LARGE?
What is $g(x)$? Ray Tracing
What is $g(x)$? Fluid flow
The inference problem

\[ y \sim N(g(x, \theta), \Sigma_y(\theta)) \]

\[ x \sim N(\mu(\theta), \Sigma_x(\theta)) \]

\[ \theta \sim \pi(\theta) \]

\[ \hat{x} \approx \arg \min_x (y - g(x))^T \Sigma_y^{-1} (y - g(x)) + (x - \mu)^T \Sigma_x^{-1} (x - \mu) \]

**How do we do this when the dimensions of \( x \) and \( y \) are huge?**
Step 1: Modelling

The objective function involves computing things like $u^T \Sigma_x^{-1} u$, which, for a dense matrix, is impossible to compute. (i.e. for a big problem it requires too much time and memory).

**Solution:** Gaussian Markov random fields. Model $Q_x = \Sigma_x^{-1}$ directly as a sparse matrix.

*This makes the computation possible.*
Gaussian Markov Random Fields (GMRFs)

What is a GMRF?
- Multivariate normal distribution
- Precision matrix $Q = \Sigma^{-1}$ is sparse
- Conditional independence properties gives sparseness

$$x_i \perp x_j \mid \mathbf{x}_{-ij} \iff Q_{ij} = 0$$
Gaussian Markov Random Fields (GMRFs)

More facts about GMRFs

— Used extensively in computational statistics (since the 80s).
— On lattices, they are very easy to compute and have well understood properties.
— Watch out for boundaries!
— From a modelling perspective they are less flexible than GRF models. From a computational point of view, you can’t do anything with GRF models.
An aside: A link to inverse problems

If you chose your GMRF carefully\(^1\), the MAP estimate for this model corresponds to the Tikhonov regularisation of a weighted nonlinear least squares problem\(^2\).

---


Be careful!

Fig. 6 Optimal kriging prediction and predictions using the DB3 basis, the Convolution-fields, and a tapered covariance. The first order spline estimate have been excluded since its estimate was indistinguishable from the DB3 estimate.

If the number of basis functions and the tapering range had been chosen such that the total computation time in Table 1 were equal for the different methods, S1 would have been most accurate, even for short ranges. We therefore conclude that the wavelet Markov approximation using the S1 basis is to prefer over the other methods compared in this section.

4.6 Discussion

One might argue that the comparison in the previous section is unfair since the methods only are compared using a fixed tapering range and a fixed tapering function. The tapering range was chosen such that the computation time for the matrix inverse is similar to the other.

Part 2: Practical computation

For the remainder of the talk, we’ll focus on ways of exploiting the sparse structure of the precision matrix in inference.

— The focus will be on the problem that was described above.
— Everything generalises to more complicated problems, including
  • Sampling algorithms (for MCMC),
  • Determinant computation (for inferring the parameters)
  • Calculating the marginal variances of a posterior GMRF (i.e. the diagonal elements of $Q_{x|y}^{-1}$)
— These methods are designed for large problems and are suitable for parallel computing (although I won’t mention that).
Outline

Introduction

Newton’s Method

Solving linear systems with Krylov subspaces

Conclusion

Extras
Newton’s method: the basic optimisation scheme

C. T. Kelley, Iterative Methods for Optimization
Newton’s method: the basic optimisation scheme

The problem

Find \( x = \text{arg min}_x f(x) \) for some function that takes a vector \( x \). In our case

\[
f(x) = (y - g(x))^T \Sigma_y^{-1} (y - g(x)) + (x - \mu)^T \Sigma_x^{-1} (x - \mu)
\]

Mathematically, we want to solve \( \nabla f(x) = 0 \).
Newton’s method: the basic optimisation scheme

The problem

Solve $\nabla f(x) = 0$.

**Newton’s Method:** Construct approximations to the minimum by iterating $x^{(k+1)} = x^{(k)} + \delta x^{(k)}$, where

$$H(x^{(k)})\delta x^{(k)} = -\nabla f(x^{(k)}).$$

Here $[H(x^{(k)})]_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$ is the *Hessian* matrix.
Computing the gradient and the Hessian

How can we compute the Hessian

— Analytical expressions are good if you have them! Even just for the gradient.
— Automatic differentiation is sometimes possible.
— Realistically, we have finite differences. This only requires the evaluation of $f(x)$.

Fact: Accurate gradients are more important that accurate Hessians!
The gradient for the example

With \( f(x) = f(x) = (y - g(x))^T \Sigma_y^{-1} (y - g(x)) + (x - \mu)^T \Sigma_x^{-1} (x - \mu) \), the gradient can be calculated as

\[
\frac{1}{2} \nabla f(x) = \Sigma_x^{-1} x + Dg(x)^T \Sigma_y^{-1} g(x) - Dg(x)^T \Sigma_y^{-1} y + \Sigma_x^{-1} x.
\]
Central difference approximation to the gradient

We define the approximate gradient $\nabla_h f(x)$ component by component as

$$
(\nabla_h f(x))_i = \frac{f(x + h e_i) - f(x - h e_i)}{2h},
$$

where the step size is chosen as $h = O(\epsilon_m^{1/3}) \approx 10^{-6}$. The error in the approximation is $O(\epsilon_m^{2/3}) \approx 10^{-11}$.
Finite difference approximation to the Hessian

We define the approximate Hessian column by column. To do this, we define for any vector $w \neq 0$

$$D^2_h f(x : w) = \frac{\nabla_h f(x + hw/ \|w\|) - \nabla_h f(x)}{h/ \|w\|},$$

where $h = \mathcal{O}(\epsilon_m^{1/3}) \approx 10^{-6}$.

The approximation to the full Hessian is $H_h = (A_h + A_h^T)/2$, where the $i$th column of $A_h$ is $D^2_h f(x : e_i)$. 
Important point

This only works if you can compute $f(x)$ cheaply.
Does an inaccurate Hessian matter?

The single most surprising fact about Newton methods

No matter how bad your Hessian approximation is, the Newton method will converge.

As a general rule, we require our approximate solution to satisfy

\[ \|H(x^{(k)})\delta x^{(k)} + \nabla f(x^{(k)})\| \leq \eta_k \|\nabla f(x^{(k)})\| \]

We usually take \( \eta_k \propto \|\nabla f(x^{(k)})\|^2 \).
The inexact Newton method

**Data:** Function $f(x)$, Initial guess $x^{(0)}$, Relative tolerance $\tau_r$, Absolute tolerance $\tau_a$.

**Result:** An approximate minimiser of $f(x)$

$r = r_0 = \|\nabla f(x^{(0)})\|

Set $x = x_0$

**while** $\|\nabla f(x)\| > \tau_r r_0 + \tau_a$ **do**

Select $\eta = O(r^2)$

Calculate $\delta x$ such that $\|H(x)\delta x + \nabla f(x)\| \leq \eta k \|\nabla f(x)\|

Set $x = x + \lambda \delta x$

**end**
Further issues

Of course, this is not sophisticated enough for real code. The most pressing problem is that we can’t guarantee that convergence will occur for any starting value.

— Line searching is the simplest way to ‘globalise’ a Newton method and it works well.
— Trust regions work better.
Outline

Introduction

Newton’s Method

Solving linear systems with Krylov subspaces

Conclusion

Extras
We really don’t want to compute a Hessian!

It’s expensive and it requires a lot of storage!
Iterative methods for solving linear systems

The inexact condition suggests that we want to be able to control how well we solve the update equation: we don’t want to waste computational resources!
The Conjugate Gradient Method

The update condition

\[ \| H(x^{(k)})\delta x^{(k)} + \nabla f(x^{(k)}) \| \leq \eta_k \| \nabla f(x^{(k)}) \| \]

suggest that we want to make \(^3\)

\[ \| H\delta x - g \| \]

small.

\(^3\)We are dropping all of the \( k \) superscripts and setting \( g \equiv -\nabla f(x^{(k)}) \).
The Conjugate Gradient Method
Choosing $p_j$ in $\delta x_{j+1} = \delta x_j + \alpha_j p_j$

— We don’t want to go in the same direction twice

$$p_i^T H p_j = 0, \quad i \neq j.$$  

— We want to use our initial ‘guess’ $\delta x_0$:

$$p_0 = \frac{r_0}{\sqrt{r_0^T H r_0}},$$

where $r_0 = g - H \delta x_0$.

— We want each new direction $p_{j+1}$ to be based on $r_j$.  

---

www.ntnu.no  
NTNU  
Norwegian University of Science and Technology  
D. Simpson, Numerical Linear Algebra
This is enough for an algorithm!

**Data:** Hessian $H$, Negative gradient $g$, Stopping parameter $\eta$.

**Result:** An approximation to $\delta x$ that satisfies the convergence condition

$r = g$, $\rho_0 = \|r^2\|$, $k = 1$, $\delta x = 0$

while $\sqrt{\rho_{k-1}} > \eta \|g\|$ do

  if $k = 1$ then
  | $p = r$
  else
  | $\beta = \rho_{k-1}/\rho_{k-2}$, $r = \beta p$
  end

  $w = Hp$
  if $p^Tw \leq 0$ then
  | **STOP**
  end

  $\alpha = \rho_{k-1}/p^Tw$
  $\delta x = \delta x + \alpha p$
  $r = r - \alpha w$, $\rho_k = \|r\|^2$

  $k = k + 1$

end
A Hessian Free Newton-CG method

Data: Function $f(x)$, Stopping parameter $\eta$.

Result: An approximation to $\delta x$ that satisfies the convergence condition

$r = -\nabla_h f(x)$, $\rho_0 = \|r^2\|$, $k = 1$, $\delta x = 0$

while $\sqrt{\rho_{k-1}} > \eta \|\nabla_h f(x)\|$ do

if $k = 1$ then
  $p = r$
else
  $\beta = \rho_{k-1}/\rho_{k-2}$, $p = r + \beta p$
end

$w = D^2_h f(x : p)$
if $p^T w \leq 0$ then
  STOP!
end

$\alpha = \rho_{k-1}/p^T w$
$\delta x = \delta x + \alpha p$
$r = r - \alpha w$, $\rho_k = \|r\|^2$
$k = k + 1$
Aside: Krylov subspaces

At the $m$th step of the algorithm, the approximation lies in the space

$$\text{span}(p_0, p_1, \ldots, p_{m-1}) = \text{span}(g, Hg, \ldots, H^{m-1}g) \equiv \mathcal{K}_m(H, g).$$

This is the $m$th Krylov subspace generated by $H$ and $g$ and it’s useful for a whole bunch of other things.
Aside: Krylov subspaces

At the $m$th step of the algorithm, the approximation lies in the space

$$\text{span}(p_0, p_1, \ldots, p_{m-1}) = \text{span}(g, Hg, \ldots, H^{m-1}g) \equiv \mathcal{K}_m(H, g).$$

This is the $m$th Krylov subspace generated by $H$ and $g$ and it’s useful for a whole bunch of other things.

— Partial least squares.
Aside: Krylov subspaces

At the $m$th step of the algorithm, the approximation lies in the space

$$\text{span}(p_0, p_1, \ldots, p_{m-1}) = \text{span}(g, Hg, \ldots, H^{m-1}g) \equiv \mathcal{K}_m(H, g).$$

This is the $m$th Krylov subspace generated by $H$ and $g$ and it’s useful for a whole bunch of other things.

— Partial least squares.
— Computing a few eigenvalues, eigenvectors, or principle components.
Aside: Krylov subspaces

At the $m$th step of the algorithm, the approximation lies in the space

$$\text{span}(p_0, p_1, \ldots, p_{m-1}) = \text{span}(g, Hg, \ldots, H^{m-1}g) \equiv K_m(H, g).$$

This is the $m$th Krylov subspace generated by $H$ and $g$ and it’s useful for a whole bunch of other things.

— Partial least squares.
— Computing a few eigenvalues, eigenvectors, or principle components.
— Solving *non-symmetric* linear systems.
Aside: Krylov subspaces

At the $m$th step of the algorithm, the approximation lies in the space

$$\text{span}(p_0, p_1, \ldots, p_{m-1}) = \text{span}(g, Hg, \ldots, H^{m-1}g) \equiv K_m(H, g).$$

This is the $m$th Krylov subspace generated by $H$ and $g$ and it’s useful for a whole bunch of other things.

— Partial least squares.
— Computing a few eigenvalues, eigenvectors, or principle components.
— Solving non-symmetric linear systems.
— Fast approximate sampling from huge GMRFs.
Aside: Krylov subspaces

At the $m$th step of the algorithm, the approximation lies in the space

$$\text{span}(p_0, p_1, \ldots, p_{m-1}) = \text{span}(g, Hg, \ldots, H^{m-1}g) \equiv K_m(H, g).$$

This is the $m$th Krylov subspace generated by $H$ and $g$ and it’s useful for a whole bunch of other things.

— Partial least squares.
— Computing a few eigenvalues, eigenvectors, or principle components.
— Solving non-symmetric linear systems.
— Fast approximate sampling from huge GMRFs.
— Computing determinants, traces, elements of the inverse.
Speeding up the convergence

*Preconditioning* is a fortuitous change of variable that makes the problem easier (and therefore faster) to solve:

\[ M\delta x = u. \]

We now need to solve

\[ HM^{-1}u = g \]

Clearly if we chose \( M \) well, this will be easier to solve (best choice \( M = H \))
So how do we chose a preconditioner?

We have two requirements

- $HM^{-1} \approx I$
- $M^{-1}$ is easy to invert.

The two most popular options are multigrid and the incomplete Cholesky decomposition.
How to use the preconditioner

The clear problem with any preconditioner is that, in order to compute $M$, you need to construct $H$, which we have avoided doing.

Idea

We don’t expect the Hessian to change very much. Therefore we compute it once and use it to build a preconditioner $M = LL^T \approx H$. **We only re-compute the preconditioner when it stops being effective.**
The Preconditioned Hessian-free Newton-CG Method

Data: Function $f(x)$, Initial guess $x^{(0)}$, Relative tolerance $\tau_r$, Absolute tolerance $\tau_a$.

Result: An approximate minimiser of $f(x)$

$r = r_0 = \|\nabla f(x^{(0)})\|$ 
Set $x = x_0$ 
while $\|\nabla f(x)\| > \tau_r r_0 + \tau_a$ do 
  Select $\eta = \mathcal{O}(r^2)$ 
  If necessary, update the preconditioner $M$ 
  Use a preconditioned Hessian-free CG method to compute $\delta x$ such that 
  $\|H(x)\delta x + \nabla f(x)\| \leq \eta_k \|\nabla f(x)\|$ 
  Set $x = x + \lambda \delta x$ 
end
The preconditioned CG method

**Data:** Function $f(x)$, Preconditioner $M$, Stopping parameter $\eta$.

**Result:** An approximation to $\delta x$ that satisfies the convergence condition

$$r = -\nabla_h f(x), \quad \rho_0 = \|r^2\|, \quad k = 1, \quad \delta x = 0$$

while $\sqrt{\rho_{k-1}} > \eta \|\nabla_h f(x)\|$ do

1. $z = Mr$, $\tau_{k-1} = z^T r$

2. if $k = 1$ then
   - $p = r$
3. else
   - $\beta = \tau_{k-1}/\tau_{k-2}$, $p = z + \beta p$
4. end

5. $w = D^2_h f(x : p)$
6. if $p^T w \leq 0$ then
   - STOP!
7. end

8. $\alpha = \tau_{k-1}/p^T w$
9. $\delta x = \delta x + \alpha p$
10. $r = r - \alpha w$, $\rho_k = \|r\|^2$
11. $k = k + 1$
end
Outline

Introduction

Newton’s Method

Solving linear systems with Krylov subspaces

Conclusion

Extras
Conclusion

In contrast to traditional statistical modelling, practical problems in spatial statistics are, by and large, *computational* in nature.

— Our models need to be computationally feasible.
— To give ourselves a good chance we need to use modern techniques from numerical analysis.
— It is important that we are actually computing what we think we are computing.
Questions?

http://www.threewordphrase.com/cosmic.htm
Outline

Introduction

Newton’s Method

Solving linear systems with Krylov subspaces

Conclusion

 Extras
Sampling from a Zero Mean GMRF

Algorithm: Sampling from a GMRF

— Sample \( z \) from a random vector with \( z_i \sim \mathcal{N}(0, 1) \)
— \( x = Q^{-1/2}z \) is a sample from the zero mean GMRF with precision matrix \( Q \).
The 10 000 000 kr Question

How do we compute $Q^{-1/2}z$?

Option 1: We compute most of it

**Fact:** We can always find an orthogonal matrix $V$ such that $V^TQV = T$ is tridiagonal. Then

$$Q^{-1/2}z = VT^{-1/2}V^Tz.$$ 

**Idea:** Just take the first $m$ columns

$$Q^{-1/2}z \approx V_mT_m^{-1/2}V_m^Tz.$$
The 10 000 000 kr Question

**How do we compute** $Q^{-1/2}z$??

**Option 2:** We compute something similar

**Fact:** We can’t calculate inverse square roots of large sparse matrices, but we can solve linear systems.

**Idea:** Compute $z^{-1/2} \approx \sum_{i=1}^{p} \alpha_i (t + \beta_i)^{-1}$. Then

$$Q^{-1/2}z \approx \sum_{i=1}^{p} \alpha_i (Q + \beta_i I)^{-1}z.$$ 

We can compute both the best rational approximation and the resulting linear systems very efficiently.
## Timings 2: Timings Harder!

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$\kappa(Q)$</th>
<th>Method</th>
<th>Matrix Vector Products</th>
<th>Linear Solves</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^2$</td>
<td>$5 \cdot 10^3$</td>
<td>Lanczos</td>
<td>70</td>
<td>—</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zolotarev</td>
<td>—</td>
<td>10</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EKSM</td>
<td>9</td>
<td>10</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cholesky</td>
<td>—</td>
<td>—</td>
<td>0.06</td>
</tr>
<tr>
<td>$10^4$</td>
<td>$5 \cdot 10^5$</td>
<td>Lanczos</td>
<td>82</td>
<td>—</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zolotarev</td>
<td>—</td>
<td>14</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EKSM</td>
<td>9</td>
<td>10</td>
<td>0.36</td>
</tr>
<tr>
<td>$10^6$</td>
<td>$5 \cdot 10^7$</td>
<td>Lanczos</td>
<td>91</td>
<td>—</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zolotarev</td>
<td>—</td>
<td>17</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EKSM</td>
<td>7</td>
<td>9</td>
<td>0.33</td>
</tr>
<tr>
<td>$10^8$</td>
<td>$5 \cdot 10^9$</td>
<td>Lanczos</td>
<td>100</td>
<td>—</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zolotarev</td>
<td>—</td>
<td>23</td>
<td>1.24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EKSM</td>
<td>6</td>
<td>7</td>
<td>0.24</td>
</tr>
<tr>
<td>$10^{10}$</td>
<td>$5 \cdot 10^{11}$</td>
<td>Lanczos</td>
<td>109</td>
<td>—</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zolotarev</td>
<td>—</td>
<td>23</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EKSM</td>
<td>4</td>
<td>5</td>
<td>0.17</td>
</tr>
</tbody>
</table>
The problem with iterative methods

Iterative methods (LSQR for least squares sampling, and the matrix function methods) have one major drawback:

**They don’t compute the log-determinant!**

**Determinants are very difficult to compute!**
Idea 1: Approximate factorisations

**Concept:** Even if we don’t want to use the approximate factorisation to compute the sample, it will give a decent approximation to the determinant.

**Problem:** We have no control over the error. Furthermore, there is no way of checking how good your answer is.
Idea 2: Matrix functions (Bai et al ’96)

If the Cholesky decomposition is unavailable, a better way is to use the identity

$$\log(\det(A)) = \text{tr}(\log(A)) = \sum_{i=1}^{n} e_i^T \log(A)e_i.$$ 

Is there a cheap way to approximate $\text{tr}(\log(A))$?
A Stochastic Estimator of the Trace

Theorem (Hutchinson ’90)

Let $B \in \mathbb{R}^{n \times n}$ be a symmetric matrix with non-zero trace. Let $Z$ be the discrete random variable which takes the values $-1, 1$ each with probability $1/2$ and let $z$ be a vector of $n$ independent samples from $Z$. Then $z^T B z$ is an unbiased estimator of $\text{tr}(B)$ and $Z$ is the unique random variable amongst zero mean random variables for which $z^T B z$ is a minimum variance, unbiased estimator of $\text{tr}(B)$.

Therefore

$$\log(\text{det}(A)) = \mathbb{E} \left( z^T \log(A) z \right).$$

This can be estimated using a Monte Carlo method.
Other possibilities

— The method above can be improved for banded matrices by a careful (deterministic) choice of $z$.
— It is not clear how to extend this to general sparsity patterns—may involve $k$–nearest neighbour graphs.
— Do any of you have ideas? Functions that the determinant optimises? Anything?