Large, Sparse Linear Systems

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & & & & & \\ 1 & 2 & -2 & & & & \\ & 2 & 2 & -3 & & & \\ & & 3 & 2 & -4 & & \\ & & & 4 & 2 & -5 & \\ & & & & 5 & 2 & -6 \\ & & & & & 6 & 2 \end{bmatrix}$$

Adrien Treuille

Where do these come up?...

Inverse Problems:

$$\mathbf{f}(\mathbf{x}) = \mathbf{y} \qquad \leftarrow \text{solve}$$

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_0) + \nabla\mathbf{f}(\mathbf{x}_0)\left(\mathbf{x} - \mathbf{x}_0\right)$$

$$\mathbf{y} \approx \left(\nabla\mathbf{f}(\mathbf{x}_0)\right)^{-1}\left(\mathbf{y} - \mathbf{f}(\mathbf{x}_0)\right)$$

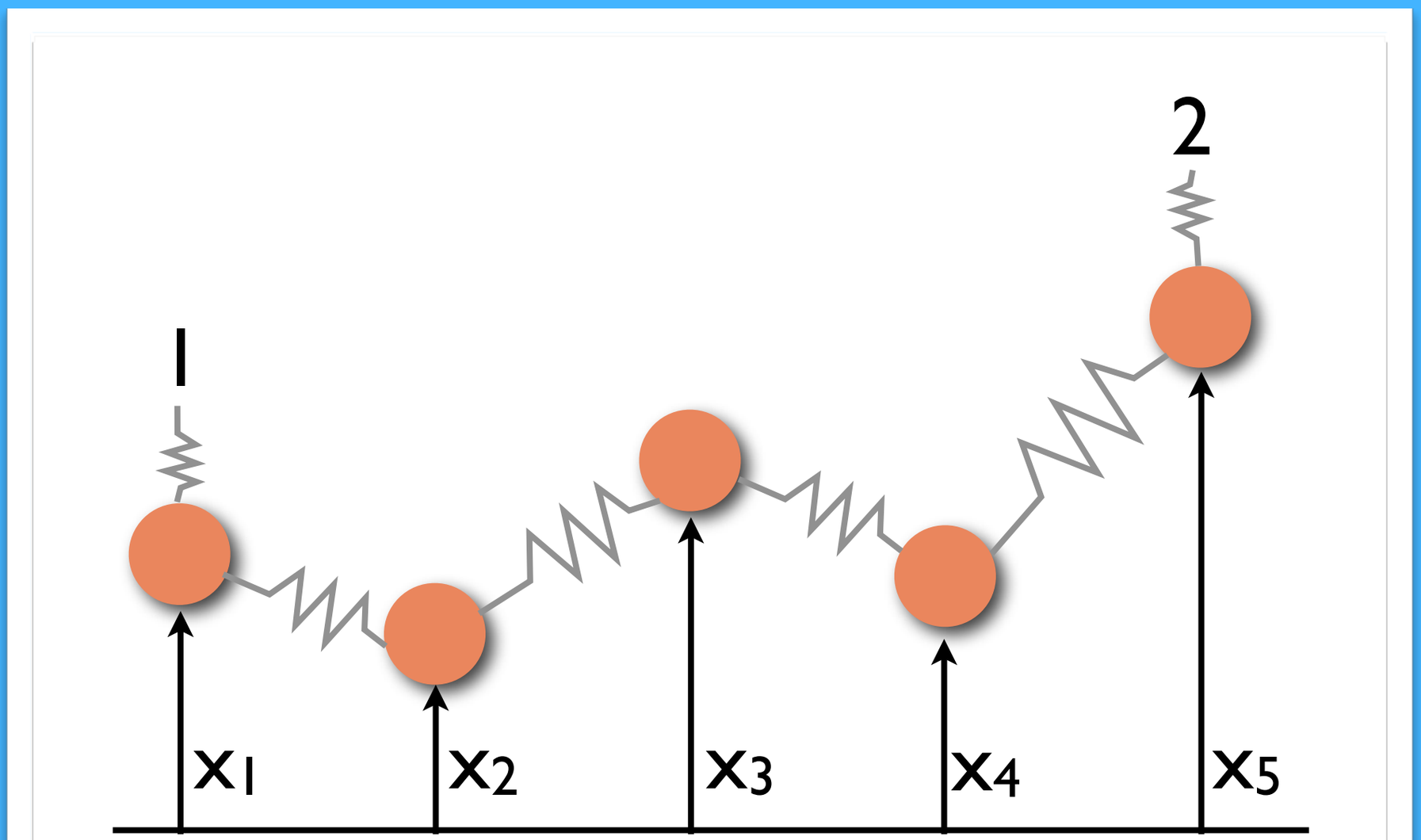
Optimization Problems:

$$\mathbf{f}(\mathbf{x}) \qquad \leftarrow \text{minimize}$$

$$\nabla\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

$$M\mathbf{x} = \mathbf{b} \qquad (\text{if } f \text{ is quadratic})$$

Example



Example

$$\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 2 \end{bmatrix}$$

How do we solve such a system?

Conjugate Gradient

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
repeat

$$\alpha_k := \frac{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}$$
$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$
if $\mathbf{r}_{k+1}$ is sufficiently small **then** exit loop **end if**

$$\beta_k := \frac{\mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}$$
$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
$k := k + 1$

Notice that we only use matrix multiplies to solve this problem!

Why does sparsity help?

Spreadsheet Example

Itereration:	0	1	2	3	4	5
Ap		5.5	0.5210686682521	-0.0176796024201	-0.3598612996713	0.3003768270818
		-6	0.6929250891795	-0.160125243602	0.5300723192475	0.032183231473
		5	-1.9330410225922	0.1703395645437	0.1039893628636	0.0107277438244
		-7	0.8881539833532	0.2391664333088	-0.3423446033796	0.032183231473
		7.5	1.087507431629	-0.0054728583811	0.2991082700936	-0.1716439011896
α		0.3189655172414	0.5186189391099	1.9502069796353	0.6430243217555	0.8034309096632
x	0	0.6379310344828	0.8489861259338	0.9318633293604	0.9583741429971	1.1666666666667
	1	0.5215517241379	0.6734258271078	0.8736591179976	1.1580803134182	1.3333333333333
	0.5	0.8189655172414	0.5522945570971	1.1821613031387	1.3836434867777	1.5
	1	0.5215517241379	0.8388473852721	1.5661501787843	1.6178256611166	1.6666666666667
	0	0.9568965517241	1.397545357524	1.7558601509734	1.8778648383937	1.8333333333333
Ax	-1					
	1.5					
	-1					
	1.5					
	-1					
r	2	0.2456896551724	-0.0245464247599	0.0099324592769	0.2413320274241	-9.99201E-16
	-1.5	0.4137931034483	0.0544290288154	0.3667063965038	0.0258570029383	1.41206E-15
	1	-0.5948275862069	0.4076840981857	0.0754866905046	0.0086190009794	-1.39298E-15
	-1.5	0.7327586206897	0.2721451440768	-0.1942789034565	0.0258570029383	1.39819E-15
	3	0.6077586206897	0.0437566702241	0.0544298768375	-0.1379040156709	-9.4369E-16
β		0.0806331747919	0.1647427318994	0.7364318162828	0.4346958813603	9.88712E-29
p	2	0.4069560047562	0.0424966192265	0.0412283217598	0.2592538090885	-9.99201E-16
	-1.5	0.2928433412604	0.1026728408732	0.4423179431909	0.2181307910951	1.41206E-15
	1	-0.514194411415	0.3229743061218	0.3133352453745	0.1448245416287	-1.39298E-15
	-1.5	0.6118088585018	0.3729362068267	0.0803631846945	0.060790548338	1.39819E-15
	3	0.8496581450654	0.1837316742228	0.1897357273941	-0.0554266764258	-9.4369E-16

Code Example

```
double ConjGrad(int n, implicitMatrix *A, double x[], double b[],
        double epsilon, // how low should we go?
        int    *steps)
{
  int       i, iMax;
  double    alpha, beta, rSqrLen, rSqrLenOld, u;

  double *r = (double *) malloc(sizeof(double) * n);
  double *d = (double *) malloc(sizeof(double) * n);
  double *t = (double *) malloc(sizeof(double) * n);
  double *temp = (double *) malloc(sizeof(double) * n);

  vecAssign(n, x, b);

  vecAssign(n, r, b);
  A->matVecMult(x, temp);
  vecDiffEqual(n, r, temp);

  rSqrLen = vecSqrLen(n, r);

  vecAssign(n, d, r);

  i = 0;
  if (*steps)
    iMax = *steps;
  else
    iMax = MAX_STEPS;

  if (rSqrLen > epsilon)
    while (i < iMax) {
      i++;
      A->matVecMult(d, t);
      u = vecDot(n, d, t);

      if (u == 0) {
    printf("(SolveConjGrad) d'Ad = 0\n");
    break;
      }

      // How far should we go?
      alpha = rSqrLen / u;
```

```
      // How far should we go?
      alpha = rSqrLen / u;

      // Take a step along direction d
      vecAssign(n, temp, d);
      vecTimesScalar(n, temp, alpha);
      vecAddEqual(n, x, temp);

      if (i & 0x3F) {
    vecAssign(n, temp, t);
    vecTimesScalar(n, temp, alpha);
    vecDiffEqual(n, r, temp);
      } else {
    // For stability, correct r every 64th iteration
    vecAssign(n, r, b);
    A->matVecMult(x, temp);
    vecDiffEqual(n, r, temp);
      }

      rSqrLenOld = rSqrLen;
      rSqrLen = vecSqrLen(n, r);

      // Converged! Let's get out of here
      if (rSqrLen <= epsilon)
    break;

      // Change direction: d = r + beta * d
      beta = rSqrLen/rSqrLenOld;
      vecTimesScalar(n, d, beta);
      vecAddEqual(n, d, r);
    }

  // free memory

  free(r);
  free(d);
  free(t);
  free(temp);

  *steps = i;
  return(rSqrLen);
```

Conjugate Gradient

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
repeat

$$\alpha_k := \frac{\mathbf{r}_k^{\mathrm{T}} \mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}} \mathbf{A} \mathbf{p}_k}$$

$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$
if $\mathbf{r}_{k+1}$ is sufficiently small **then** exit loop **end if**

$$\beta_k := \frac{\mathbf{r}_{k+1}^{\mathrm{T}} \mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathrm{T}} \mathbf{r}_k}$$

$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
$k := k + 1$

Questions