Problem 1

This is the AMPL code.

```
set POINTS;

param PX {POINTS};
param PY {POINTS};

var a;
var b;
var c;
var delta;

var abs_dist {POINTS};
var max_distance;

minimize distance: max_distance;


subject to have_line:
   a + b >= 5;

subject to dist1 {p in POINTS}:
   abs_dist[p] >= a * PX[p] + b * PY[p] - c;

subject to dist2 {p in POINTS}:
   abs_dist[p] >= -a * PX[p] - b * PY[p] + c;

subject to mdist {p in POINTS}:
   max_distance >= abs_dist[p];

data;

set POINTS := P1, P2, P3, P4, P5, P6, P7;

param: PX PY :=
P1     1  3
P2     2  5
P3     3  7
P4     5  11
P5     7  14
P6     8  15
P7     10 19;
```

The variable abs_dist contains the absolute distance value from the the line for every point in POINTS. Constraints dist1 and dist2 ensure it always contains at least the maximum of the distance and its negation. Since we are minimizing the distance, the solver will never choose a bigger distance.

Constraint mdist ensures that the variable max_distance contains the biggest absolute distance value for all points in POINTS.

The constraint have_line ensures that the linear equation actually describes a line. Otherwise, the solver would find the minimal objective value at a=b=c. Number “5” is just an arbitrary positive number. By adding this constraint, the feasible region is modified in such a way, that the equations “y=-a” and “x=-a” (where a>0) are no longer allowed (horizontal or vertical line in the negative part of the coordinate system). By looking at the points (they are all in the sector where x>0 and y>0), we can see that such a line will not minimize the problem (a < 0 and b < 0 can be achieved by multiplying the whole equation by -1, therefore resulting in a>0 and b>0).

```
MINOS 5.51: optimal solution found.
6 iterations, objective 4
:        _varname      _var    :=
1    a                   12
2    b                   -7
3    c                  -13
4    "abs_dist['P1']"     4
5    "abs_dist['P2']"     4
6    "abs_dist['P3']"     4
7    "abs_dist['P4']"     4
8    "abs_dist['P5']"     4
9    "abs_dist['P6']"     4
10   "abs_dist['P7']"     4
11   max_distance         4
;
```

Therefore, the line with a=12, b=-7, c=-13 minimizes the maximal error. Note, that max_distance is not the actual maximal distance, since the constraint a+b>5 scales the problem. i.e. a+b>10 would result in greater numbers of a, b and c. The proportions, however, stay the same, therefore resulting in the same line. It would be the actual (geometric) distance for “sqrt(a^2 + b^2) = 1” (Hesse Form).

Problem 4

```
var flowSA >= 0, <= 6;
var flowSC >= 0, <= 10;
var flowSB >= 0, <= 1;
var flowAD >= 0, <= 4;
var flowAE >= 0, <= 1;
var flowBE >= 0, <= 20;
var flowAB >= 0, <= 2;
var flowCB >= 0, <= 2;
var flowCF >= 0, <= 5;
var flowDE >= 0, <= 2;
var flowEF >= 0, <= 6;
var flowDG >= 0, <= 5;
var flowEG >= 0, <= 10;
var flowFT >= 0, <= 4;
var flowGT >= 0, <= 12;

maximize flow:
  flowGT + flowFT;

subject to nodeA:
  flowSA = flowAB + flowAD + flowAE;

subject to nodeB:
  flowAB + flowSB + flowCB = flowBE;

subject to nodeC:
  flowSC = flowCB + flowCF;

subject to nodeD:
  flowAD = flowDG + flowDE;

subject to nodeE:
  flowBE + flowAE + flowDE = flowEG + flowEF;

subject to nodeF:
  flowCF + flowEF = flowFT;

subject to nodeG:
  flowDG + flowEG = flowGT;
```

We create a variable for the flow from nodes to nodes – for every edge that is shown in the graph. The constraints say that no edge may have a higher flow than the corresponding capacity.

AMPL find a maximum with a object function value of 13 and the following variable bindings.

```
1    flowSA   6
2    flowSC   6
3    flowSB   1
4    flowAD   4
5    flowAE   1
6    flowBE   4
7    flowAB   1
8    flowCB   2
9    flowCF   4
10   flowDE   0
11   flowEF   0
12   flowDG   4
13   flowEG   5
14   flowFT   4
15   flowGT   9
```

Problem 3

```
param MACHINES > 0;

param pos_x {1..MACHINES};
param pos_y {1..MACHINES};

var dist_x {1..MACHINES};
var dist_y {1..MACHINES};

var x;
var y;

minimize overall_distance: sum {m in 1..MACHINES} (dist_x[m] + dist_y[m]);

subject to distance_x_1 {m in 1..MACHINES}:
  dist_x[m] >= x - pos_x[m];

subject to distance_x_2 {m in 1..MACHINES}:
  dist_x[m] >= -x + pos_x[m];

subject to distance_y_1 {m in 1..MACHINES}:
  dist_y[m] >= y - pos_y[m];

subject to distance_y_2 {m in 1..MACHINES}:
  dist_y[m] >= -y + pos_y[m];


data;

param MACHINES := 4;

param pos_x := 1 3  2 0  3 -2  4 1;
param pos_y := 1 0  2 -3  3 1  4 4;
```

Solution

```
MINOS 5.51: optimal solution found.
2 iterations, objective 14
:     _varname   _var    :=
1    'dist_x[1]'   2
2    'dist_x[2]'   1
3    'dist_x[3]'   3
4    'dist_x[4]'   0
5    'dist_y[1]'   1
6    'dist_y[2]'   4
7    'dist_y[3]'   0
8    'dist_y[4]'   3
9    x             1
10   y             1
;
```

Problem 2

```
var workers1 >= 0;
var workers2 >= 0;
var workers3 >= 0;
var workers4 >= 0;

var employ1 >= 0;
var employ2 >= 0;
```

```
var q1produce >= 0;
var q2produce >= 0;
var q3produce >= 0;
var q4produce >= 0;

var inventory1 >= 0;
var inventory2 >= 0;
var inventory3 >= 0;
var inventory4 >= 0;

minimize cost: 500 * workers1 + 500 * workers2 + 500 * workers3 + 500 * workers4 + 50 * inventory1 + 50 *
inventory2 + 50 * inventory3 + 50 * inventory4;

subject to workers1a:
  workers1 = employ1;

subject to workers2a:
  workers2 = employ1 + employ2;

subject to workers3a:
  workers3 = employ1 + employ2;

subject to workers4a:
  workers4 = employ2;

subject to inventory1a:
  inventory1 = q1produce - 600;

subject to inventory2a:
  inventory2 = inventory1 + q2produce - 300;

subject to inventory3a:
  inventory3 = inventory2 + q3produce - 800;

subject to inventory4a:
  inventory4 = inventory3 + q4produce - 100;

subject to inventory4empty:
  inventory4 = 0;

subject to q1producea:
  q1produce <= 50 * workers1;

subject to q2producea:
  q2produce <= 50 * workers2;

subject to q3producea:
  q3produce <= 50 * workers3;

subject to q4producea:
  q4produce <= 50 * workers4;
```

Solution

```
MINOS 5.51: optimal solution found.
2 iterations, objective 24000
:     _varname   _var    :=
1    workers1      14
2    workers2      16
3    workers3      16
4    workers4       2
5    employ1       14
6    employ2        2
7    q1produce    600
8    q2produce    300
```

```
9    q3produce    800
10   q4produce    100
11   inventory1     0
12   inventory2     0
13   inventory3     0
14   inventory4     0
;
```

I assume that workers do not necessarily have to produce exactly 50 shoes per quarter. They can also produce less (it would not make sense to produce more shoes than we can sell).

The variables for “workers” contain the number of workers per quarter. The “employ” variables contain the number of workers that we hire per quarter. We cannot hire anybody in the third or fourth quarter, since we only have four quarters and workers have to be employed for three quarters.

As we can see from the solution, we do not need the inventory at all.