Applied Convex Models

Nick Henderson, AJ Friend (Stanford University)
Kevin Carlberg (Meta)

August 8, 2022

Image in-painting (`inpaint.ipynb`)

Outline

Image in-painting (`inpaint.ipynb`)

Kalman filtering (`robust_kalman.ipynb`)

Portfolio optimization (`portfolio_optimization.ipynb`)

Nonnegative matrix factorization (`nonneg_matrix_fact.ipynb`)

Optimal advertising (`optimal_advertising.ipynb`)

Image in-painting

Original

Corrupted

Image in-painting

guess pixel values in obscured/corrupted parts of image

- **decision variable** $x \in \mathbf{R}^{m \times n \times 3}$
- $x_{i,j} \in [0,1]^3$ gives RGB values of pixel (i,j)
- many pixels missing
- K: set of known pixel IDs, whose values given by **data** $y \in \mathbf{R}^{m \times n \times 3}$

total variation in-painting: choose pixel values $x_{i,j} \in \mathbf{R}^3$ to minimize

$$\mathsf{TV}(x) = \sum_{i,j} \left\| \begin{bmatrix} x_{i+1,j} - x_{i,j} \\ x_{i,j+1} - x_{i,j} \end{bmatrix} \right\|_2$$

that is, for each pixel, minimize distance to neighbors below and to the right, subject to known pixel values

In-painting: Convex model

$$
\begin{array}{ll}
\text{minimize} & \mathrm{TV}(x) \\
\text{subject to} & x_{i,j} = y_{i,j} \text{ if } (i,j) \in K
\end{array}
$$

In-painting: Code example

```
# K[i, j] == 1 if pixel value known, 0 if unknown
from cvxpy import *
variables = []
constr = []
for i in range(3):
    x = Variable(rows, cols)
    variables += [x]
    constr += [multiply(K, x - y[:,:,i]) == 0]

prob = Problem(Minimize(tv(*variables)), constr)
prob.solve(solver=SCS)
```

In-painting: 600×512 color image; about 900k variables

Original

Corrupted

Image in-painting (`inpaint.ipynb`)

In-painting

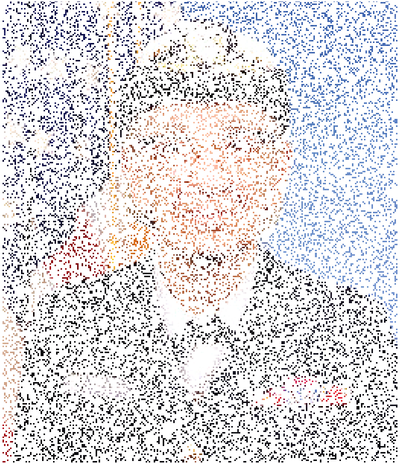
Original

Recovered

In-painting (80% of pixels removed)

Original

Corrupted



In-painting (80% of pixels removed)

Original

Recovered

Kalman filtering (`robust_kalman.ipynb`)

Outline

Image in-painting (`inpaint.ipynb`)

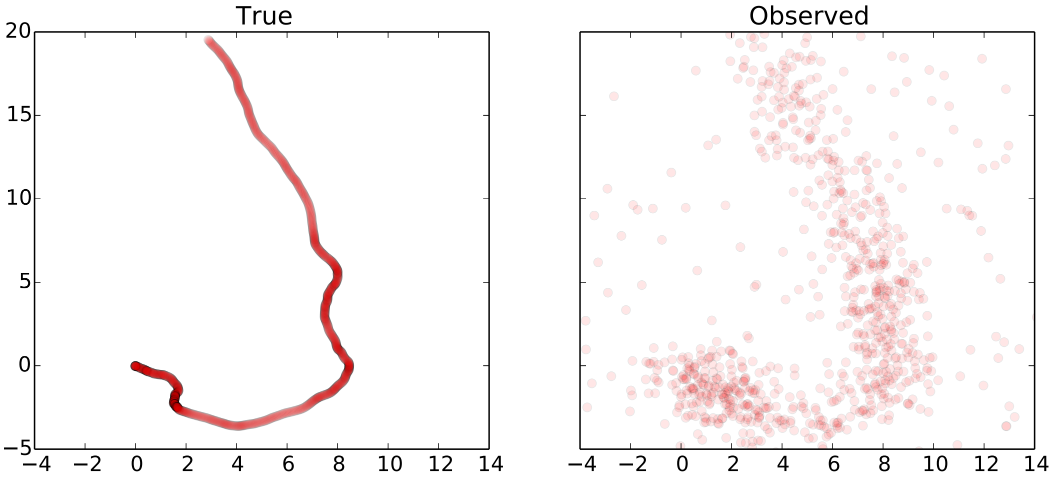
Kalman filtering (`robust_kalman.ipynb`)

Portfolio optimization (`portfolio_optimization.ipynb`)

Nonnegative matrix factorization (`nonneg_matrix_fact.ipynb`)

Optimal advertising (`optimal_advertising.ipynb`)

Vehicle tracking



Kalman filtering

- estimate vehicle path from noisy position measurements (with outliers)
- dynamic model of vehicle state x_t:

$$x_{t+1} = Ax_t + Bw_t, \quad y_t = Cx_t + v_t$$

- **Given**:
  - A, B: matrices characterizing time-discrete dynamics
  - C: output-measurement matrix
  - y_t, $t = 1, \ldots, N$: position measurements over N time steps
- **Unknown**:
  - x_t: vehicle state (position, velocity): **to be estimated**
  - w_t: unknown drive force on vehicle
  - v_t: noise

Kalman filter and Robust Kalman filter

Kalman filter:

- estimate x_t by solving

$$\begin{array}{ll} \text{minimize} & \sum_{t=1}^{N} \left(\|w_t\|_2^2 + \gamma\|v_t\|_2^2\right) \\ \text{subject to} & x_{t+1} = Ax_t + Bw_t, \quad y_t = Cx_t + v_t, \quad t = 1, \ldots, N \end{array}$$

  - can interpret w_t and v_t as the **residuals** of the equations
- a least-squares problem; maximum likelihood if assuming w_t, v_t Gaussian

Robust Kalman filter:

- to handle outliers in v_t, replace square cost with Huber cost ϕ

$$\begin{array}{ll} \text{minimize} & \sum_{t=1}^{N} \left(\|w_t\|_2^2 + \gamma\phi(v_t)\right) \\ \text{subject to} & x_{t+1} = Ax_t + Bw_t, \quad y_t = Cx_t + v_t, \quad t = 1, \ldots, N \end{array}$$

- No longer least squares due to Huber cost

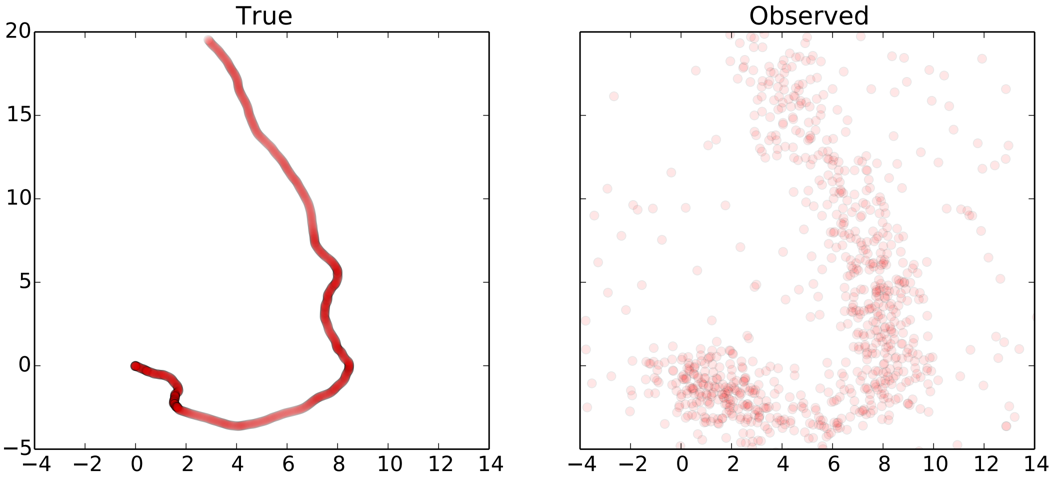
Robust KF CVXPY code

```
from cvxpy import *
x = Variable(4,n+1)
w = Variable(2,n)
v = Variable(2,n)

obj = sum_squares(w)
obj += sum(huber(norm(v[:,t])) for t in range(n))
obj = Minimize(obj)
constr = []
for t in range(n):
    constr += [ x[:,t+1] == A*x[:,t] + B*w[:,t] ,
                y[:,t]   == C*x[:,t] + v[:,t]   ]

Problem(obj, constr).solve()
```

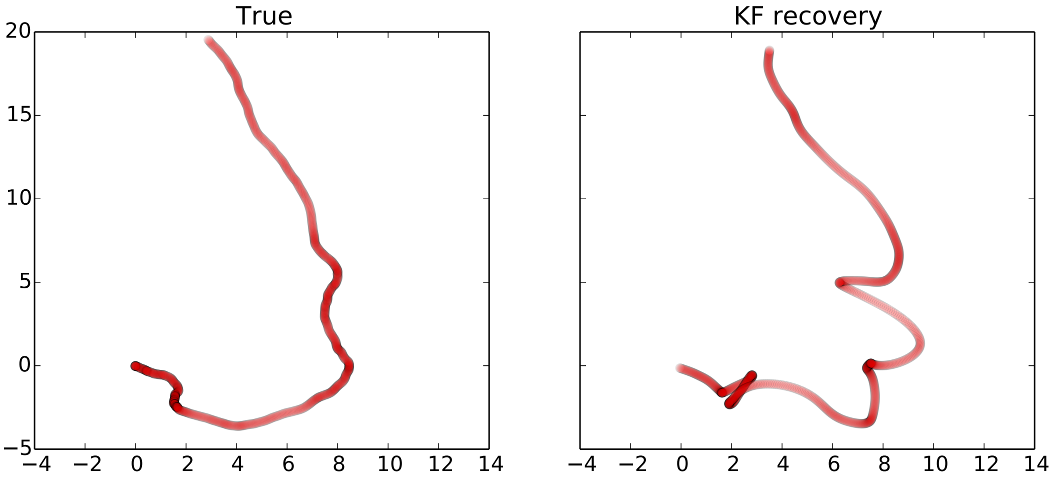
Example

- $N = 1000$ time steps
- w_t standard Gaussian
- v_t standard Gaussian, except 30% are outliers with $\sigma = 10$

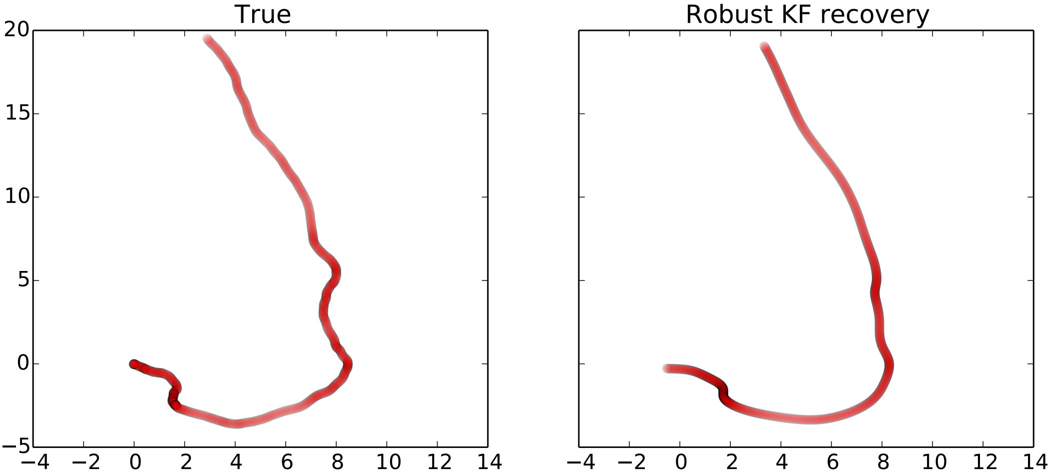
Example



Example



Example



Portfolio optimization (`portfolio_optimization.ipynb`)

Outline

Image in-painting (`inpaint.ipynb`)

Kalman filtering (`robust_kalman.ipynb`)

Portfolio optimization (`portfolio_optimization.ipynb`)

Nonnegative matrix factorization (`nonneg_matrix_fact.ipynb`)

Optimal advertising (`optimal_advertising.ipynb`)

Portfolio allocation vector

- invest fraction w_i in asset i for $i = 1, \ldots, n$
- $w \in \mathbf{R}^n$ is *portfolio allocation vector*
- $\mathbf{1}^T w = 1$
- $w_i < 0$ means *short position* in asset i (borrow shares and sell now; replace later)
- $w \geq 0$ is a *long only* portfolio
- $\|w\|_1 = \mathbf{1}^T w_+ + \mathbf{1}^T w_-$ is *leverage* (there are other definitions)
  - smaller leverage = fewer investments (sparser)

Asset Returns

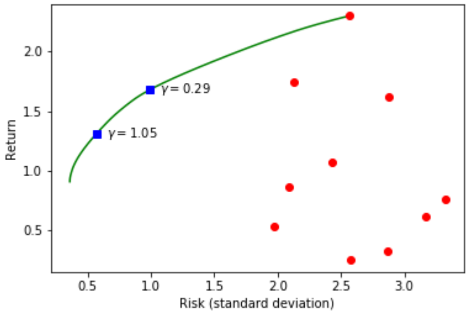
- investments held for one period
- initial prices $p_i > 0$; end of period process $p_i^+ > 0$
- asset (fractional) returns $r_i = (p_i^+ - p_i)/p_i$
- portfolio (fractional) return $R = \sum_i r_i w_i = r^T w$
- common model: r is a random variable, with mean $\mathbf{E}[r] = \mu$, covariance $\mathbf{E}[(r-\mu)(r-\mu)^T] = \Sigma$
- so R is a random variable with $\mathbf{E}[R] = \mu^T w$, $\mathbf{var}[R] = w^T \Sigma w$
- $\mathbf{E}[R]$ is (mean) return of portfolio
- $\mathbf{var}[R] = w^T \Sigma w$ is risk of portfolio
- Finance: high return, low risk (multiobjective)

Classical (Markowitz) portfolio optimization

$$\begin{array}{ll} \text{minimize} & -\mu^T w + \gamma w^T \Sigma w \\ \text{subject to} & \mathbf{1}^T w = 1, \;\; w \in \mathcal{W} \end{array}$$

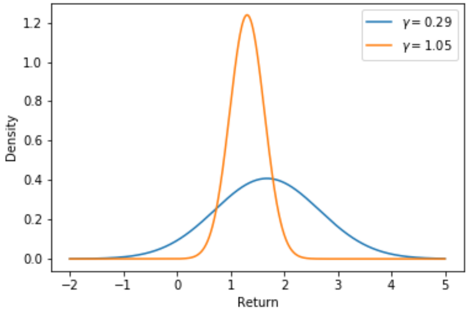
- variable $w \in \mathbf{R}^n$
- $\mathcal{W}$ is set of allowed portfolios
- common case $\mathcal{W} = \mathbf{R}^n_+$ (long only)
- $\gamma > 0$ is risk aversion parameter
- $\mu^T w - \gamma w^T \Sigma w$ is risk-adjusted return
- varying γ gives (convex hull of) Pareto-optimal risk-return trade-off
- can also fix return and minimize risk, etc.
- To limit leverage use $\|w\|_1 \leq L^{\mathsf{max}}$

Pareto front



- Pareto front shows Pareto-optimal allocations
- Red points show single-asset allocation points

Pareto front



Two Pareto-optimal portfolios:

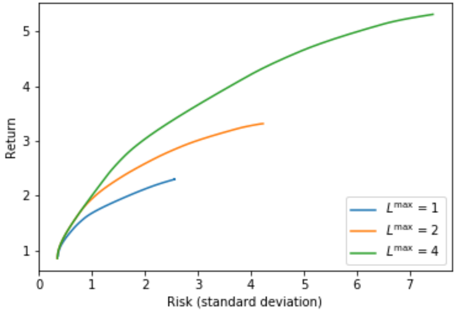
- $\gamma = 0.29$: higher return, higher risk
- $\gamma = 1.05$: lower return, lower risk

Leverage

- Now, introduce a constraint on leverage:

$$\begin{array}{ll} \text{minimize} & -\mu^T w + \gamma w^T \Sigma w \\ \text{subject to} & \mathbf{1}^T w = 1, \;\; w \in \mathcal{W} \\ & \|w\|_1 \leq L_{\max} \end{array}$$

Pareto curves for different values of $L_{\max}$:



- Larger values of $L_{\max}$ are less restrictive and enable superior portfolios in terms of risk and return

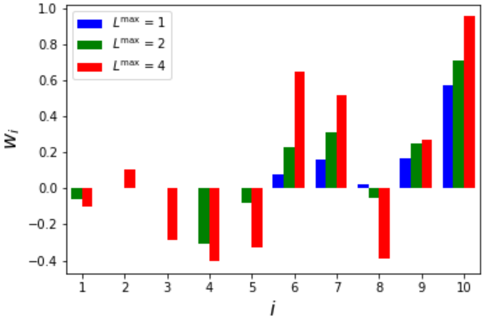
Leverage

- Now, introduce a constraint on risk:

$$\begin{array}{ll} \text{minimize} & -\mu^T w \\ \text{subject to} & \mathbf{1}^T w = 1, \;\; w \in \mathcal{W} \\ & \|w\|_1 \leq L_{\max} \\ & w^T \Sigma w \leq 2 \end{array}$$

  - Single objective

Portfolios for different values of $L_{\max}$:



- Smaller values of $L_{\max}$ enforce sparsity and smaller variation in the resulting portfolios (lower leverage)

Nonnegative matrix factorization (`nonneg_matrix_fact.ipynb`)

Outline

- Image in-painting (`inpaint.ipynb`)
- Kalman filtering (`robust_kalman.ipynb`)
- Portfolio optimization (`portfolio_optimization.ipynb`)
- Nonnegative matrix factorization (`nonneg_matrix_fact.ipynb`)
- Optimal advertising (`optimal_advertising.ipynb`)

Nonnegative matrix factorization

- **goal**: factor $A \in \mathbf{R}_+^{m \times n}$ such that
$$A \approx WH,$$
where $W \in \mathbf{R}_+^{m \times k}$, $H \in \mathbf{R}_+^{k \times n}$ and $k \ll n, m$
- W, H give nonnegative low-rank approximation to A
- low-rank means data more interpretable as combination of just k features
- nonegativity may be natural to the data, e.g., no negative words in a document
- applications in recommendation systems, signal processing, clustering, computer vision, natural language processing

NMF formulation

- many ways to formalize $A \approx WH$
- for given A and k, we'll try to find W and H that solve

$$\begin{array}{ll} \text{minimize}_{W,H} & \|A - WH\|_F^2 \\ \text{subject to} & W_{ij} \geq 0 \\ & H_{ij} \geq 0 \end{array}$$

- $\|X\|_F = \sqrt{\sum_{ij} X_{ij}^2}$ is the matrix **Frobenius norm**

Principal component analysis

- NMF can be thought of as a dimensionality reduction technique
- PCA is a related dimensionality reduction method, solving the problem

$$\text{minimize}_{W,H} \quad \|A - WH\|_F^2$$

  for $W \in \mathbf{R}_+^{m\times k}$, $H \in \mathbf{R}_+^{k\times n}$, without nonnegativity constraint
- PCA has "analytical" solution via the **singular value decomposition**
- won't go further into the interpretation of the models; focus on methods for computing NMF instead

Biconvexity

- the NMF problem

$$
\begin{array}{ll}
\text{minimize}_{W,H} & \|A - WH\|_F^2 \\
\text{subject to} & W_{ij} \geq 0 \\
& H_{ij} \geq 0
\end{array}
$$

  is **nonconvex** due to the product WH
- however, the objective function is **biconvex**: convex in either W or H if we hold the other fixed

Alternating minimization

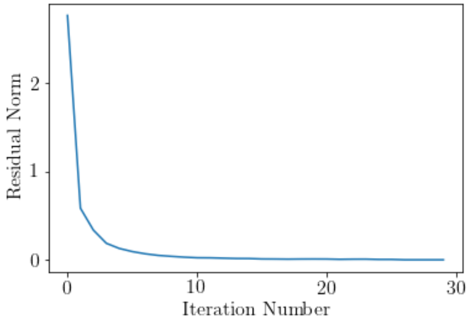
biconvexity suggests the following algorithm:

- initialize W^0
- for $k = 0, 1, 2, \ldots$
-
$$\begin{array}{rll} H^{k+1} = & \text{argmin}_H & \|A - W^k H\|_F^2 \\ & \text{subject to} & H_{ij} \geq 0 \end{array}$$
-
$$\begin{array}{rll} W^{k+1} = & \text{argmin}_W & \|A - W H^{k+1}\|_F^2 \\ & \text{subject to} & W_{ij} \geq 0 \end{array}$$

In CVXPY

```
for iter_num in range(1, 1+MAX_ITERS):
    # For odd iterations, treat Y constant, optimize over X.
    if iter_num % 2 == 1:
        X = cvx.Variable(k, n)
        constraint = [X >= 0]
    # For even iterations, treat X constant, optimize over Y.
    else:
        Y = cvx.Variable(m, k)
        constraint = [Y >= 0]

    # Solve the problem.
    obj = cvx.Minimize(cvx.norm(A - Y*X, 'fro'))
    prob = cvx.Problem(obj, constraint)
    prob.solve(solver=cvx.SCS)
```

NMF results in CVXPY



- Residual goes to zero

Discussion

- expression $A - W^k H$ is **linear** in variable H
- $\|A - W^k H\|_F^2$ is exactly the least squares objective, but with matrix instead of vector variable
- each subproblem is a convex nonnegative least squares problem
- no guarantee of global minimum, but we do get a local minimum
- due to biconvexity, the objective function **decreases** at each iteration, meaning that the iteration converges

Extensions

sparse factors with ℓ_1 penalty

$$
\begin{array}{ll}
\text{minimize}_{W,H} & \|A - WH\|_F^2 + \sum_{ij} \left(|W_{ij}| + |H_{ij}|\right) \\
\text{subject to} & W_{ij} \geq 0 \\
& H_{ij} \geq 0
\end{array}
$$

Extensions

matrix completion: only observe subet of entries A_{ij} for $(i,j) \in \Omega$

- use low-rank assumption to estimate missing entries

$$
\begin{array}{ll}
\text{minimize}_{W,H,Z} & \sum_{i,j\in\Omega}(A_{ij} - Z_{ij})^2 \\
\text{subject to} & Z = WH \\
& W_{ij} \geq 0 \\
& H_{ij} \geq 0
\end{array}
$$

Optimal advertising (`optimal_advertising.ipynb`)

Outline

Ad display

- m advertisers/ads, $i = 1, \ldots, m$
- n time slots, $t = 1, \ldots, n$
- T_t is total traffic in time slot t
- $D_{it} \geq 0$ is number of ad i displayed in period t
- $\sum_i D_{it} \leq T_t$
- contracted minimum total displays: $\sum_t D_{it} \geq c_i$
- goal: choose D_{it}

Clicks and revenue

- C_{it} is number of clicks on ad i in period t
- click model: $C_{it} = P_{it} D_{it}$
- $P_{it} \in [0, 1]$: fraction of ads i in period t that are clicked
- payment: $R_i > 0$ per click for ad i, up to budget B_i
- ad revenue

  $$S_i = \min\{R_i \sum_t C_{it}, B_i\}$$

  is a concave function of D

Ad optimization

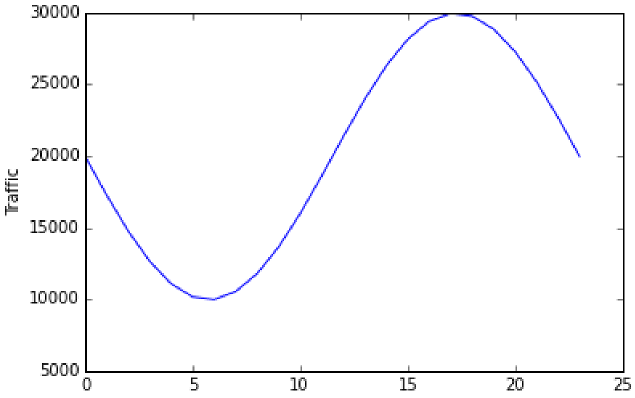
- choose displays to maximize revenue:

$$\begin{array}{ll} \text{maximize} & \sum_i S_i = \min\{R_i \sum_t P_{it} D_{it}, B_i\} \\ \text{subject to} & D \geq 0, \quad D^T \mathbf{1} \leq T, \quad D\mathbf{1} \geq c \end{array}$$

- variable is $D \in \mathbf{R}^{m \times n}$
- data are T, c, R, B, P
- constraint interpretation:
  - $D \geq 0$: non-negative number of each ad in each time period
  - $D^T\mathbf{1} \leq T$: cannot exceed total traffic in each time slot
  - $D\mathbf{1} \geq c$: cannot violate minimum number of contracted ad displays

Ad optimization example

- 24 hourly periods, 5 ads (A–E)
- total traffic T_t:



Example

- ad data:

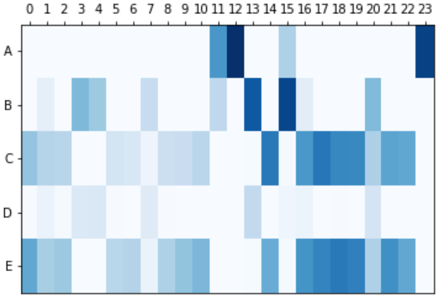
Ad	A	B	C	D	E
c_i	61000	80000	61000	23000	64000
R_i	0.15	1.18	0.57	2.08	2.43
B_i	25000	12000	12000	11000	17000

- c_i: minimum contracted amount for ad i
- R_i: payment per click for ad i
- B_i: maximum budget for ad i

Ad optimization CVXPY code

```
from cvxpy import *
D = Variable(m,n)
Si = [minimum(R[i]*P[i,:]*D[i,:].T, B[i]) for i in range(m)]
prob = Problem(Maximize(sum(Si)),
               [D >= 0,
                D.T*np.ones(m) <= T,
                D*np.ones(n) >= c])
prob.solve()
```

Ad optimization results in CVXPY



Example

- ad revenue

Ad	A	B	C	D	E
c_i	61000	80000	61000	23000	64000
R_i	0.15	1.18	0.57	2.08	2.43
B_i	25000	12000	12000	11000	17000
$\sum_t D_{it}$	61000	80000	148116	23000	167323
S_i	182	12000	12000	11000	7760

- Only show minimum number of ad A; makes very little money
- Maximize the budget for ads B, C, and D