In [2]:
```
import numpy as np
import cvxpy as cp
import matplotlib.pyplot as plt
```

A4.25) Probability Bounds

pg. 41

We can create a 4-D tensor with the axes representing X1, X2, X3, X4. Then, each entry (x1, x2, x3, x4) is equal to the joint probability of $X1 = x1, X2 = x2, X3 = x3, X4 = x4$.

In [ ]:
```
z = np.zeros((2,2,2,2))
z.sum(axis=(0, 3))[0, 1, 0]
```

Out[ ]:
```
np.float64(0.0)
```

In [22]:
```
z = np.ones((3, 1))
z.sum(axis=0)
```

Out[22]:
```
array([3.])
```

In [34]:
```
X = cp.Variable((2, 2, 2, 2))
constr425 = []

constr425 += [
    X >= 0,
    cp.sum(X) == 1,
    cp.sum(X[1, :, :, :]) == 0.9,
    cp.sum(X[:, 1, :, :]) == 0.9,
    cp.sum(X[:, :, 1, :]) == 0.1
]

# P(X1 = 1, X4 = 0 | X3 = 1) = 0.7
constr425 += [
    cp.sum(X[1, :, 1, 0]) == 0.7 * cp.sum(X[:, :, 1, :])
]

# P(X4 = 1 | X2 = 1, X3 = 0) = 0.6
constr425 += [
    cp.sum(X[:, 1, 0, 1]) == 0.6 * cp.sum(X[:, 1, 0, :])
]

prob425min = cp.Problem(cp.Minimize(cp.sum(X[:, :, :, 1])), constr425)
prob425min.solve()
print("Minimum possible value of P(X4=1): ", prob425min.value)
```

```
Minimum possible value of P(X4=1):  0.47999999984476227
```

```
C:\Users\kyler\AppData\Local\Temp\ipykernel_22044\1265068490.py:23: UserWarning: The
problem has an expression with dimension greater than 2. Defaulting to the SCIPY bac
kend for canonicalization.
  prob425min.solve()
```

In [35]:
```
prob425max = cp.Problem(cp.Maximize(cp.sum(X[:, :, :, 1])), constr425)
prob425max.solve()
print("Maximum possible value of P(X4=1): ", prob425max.value)
```

```
Maximum possible value of P(X4=1):  0.6100000000407165
```

```
C:\Users\kyler\AppData\Local\Temp\ipykernel_22044\2155099936.py:2: UserWarning: The problem has an expression with dimension greater than 2. Defaulting to the SCIPY backend for canonicalization.
  prob425max.solve()
```

A6.13) Fitting Censored Data

pg. 79

a) We can find c and $y^{M+1} \dots y^K$ that minimizes J by creating a constraint that ensures that $c^T x_i \geq D$ for $i \in [M, K]$, since we know the true value of the censored values is always greater than the upper bound D.

In [3]:
```
np.random.seed(15)

n = 20;  # dimension of x's
M = 25;  # number of non-censored data points
K = 100; # total number of points
c_true = np.random.randn(n,1)
X = np.random.randn(n,K)
y = np.dot(np.transpose(X),c_true) + 0.1*(np.sqrt(n))*np.random.randn(K,1)

# Reorder measurements, then censor
sort_ind = np.argsort(y.T)
y = np.sort(y.T)
y = y.T
X = X[:, sort_ind.T]
D = (y[M-1]+y[M])/2.0
y = y[list(range(M))]
```

In [18]:
```
X.shape, y.shape
```

Out[18]:
```
((20, 100, 1), (25, 1))
```

In [37]:
```
c = cp.Variable((n, 1))
u = cp.Variable(K-M) # add unknown value to upper bound
obj613 = cp.Minimize(cp.sum_squares(y - X[:, :M].T @ c))
constr613 = [
    X[:, M:].T @ c >= D
]
prob613 = cp.Problem(obj613, constr613)
prob613.solve()
c_hat = c.value
print("c_hat = ", c.value.T)
```

```
c_hat =  [[-0.31241271  0.44380789 -0.40245709 -0.71305005  0.34677839 -1.83550597
  -0.94876006 -1.24892195 -0.40562112 -0.42717111 -0.23496884  0.20241395
   0.5029806   0.32425101 -0.46535665  0.58949698 -0.20980902  1.61277963
   0.62789831  0.09986467]]
```

```
C:\Users\kyler\AppData\Local\Temp\ipykernel_29640\877152390.py:8: UserWarning: The p
roblem includes expressions that don't support CPP backend. Defaulting to the SCIPY
backend for canonicalization.
  prob613.solve()
```

In [38]:

```
prob613ls = cp.Problem(obj613)
prob613ls.solve()
c_hat_ls = c.value
print("c_hat_ls = ", c.value.T)
```

```
c_hat_ls =  [[-0.86949623  0.38779257 -0.07924954 -0.52689695  0.44848029 -2.1460358
  -0.79172669 -0.86627365 -0.18321402 -0.25059169 -0.15833986  0.55531191
   0.42815852  0.06903265 -0.43355176  0.35683189 -0.20242313  2.00710944
   0.81069272  0.09358954]]
```

```
C:\Users\kyler\AppData\Local\Temp\ipykernel_29640\190068961.py:2: UserWarning: The p
roblem includes expressions that don't support CPP backend. Defaulting to the SCIPY
backend for canonicalization.
  prob613ls.solve()
```

In [39]:

```
print("c_hat error: ", np.linalg.norm(c_true - c_hat, 2) / np.linalg.norm(c_true, 2
```

```
c_hat error:  0.1611548300994136
```

In [40]:

```
print("c_hat_ls error: ", np.linalg.norm(c_true - c_hat_ls, 2) / np.linalg.norm(c_t
```

```
c_hat_ls error:  0.3332218561062694
```

A15.13) Bandlimited signal recovery from zero-crossings

pg. 183

In [41]:

```
# data for problem on bandlimited signal recovery from zero crossings
import numpy as np
n = 2048
f_min = 4
B = 9
s = np.array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
y = np.array([-1.0094, -0.9767, -0.9433, -0.9090, -0.8740, -0.8383, -0.8019, -0.764
```

ooouugh i dont have time :'((

A19.10) Scheduling

pg. 264

a) We can make the objective a convex combination of C and T- for example, minimize $\theta C + (1-\theta)T$ s.t. $\theta \in [0,1]$, and then solve for various values of θ.

In [43]:

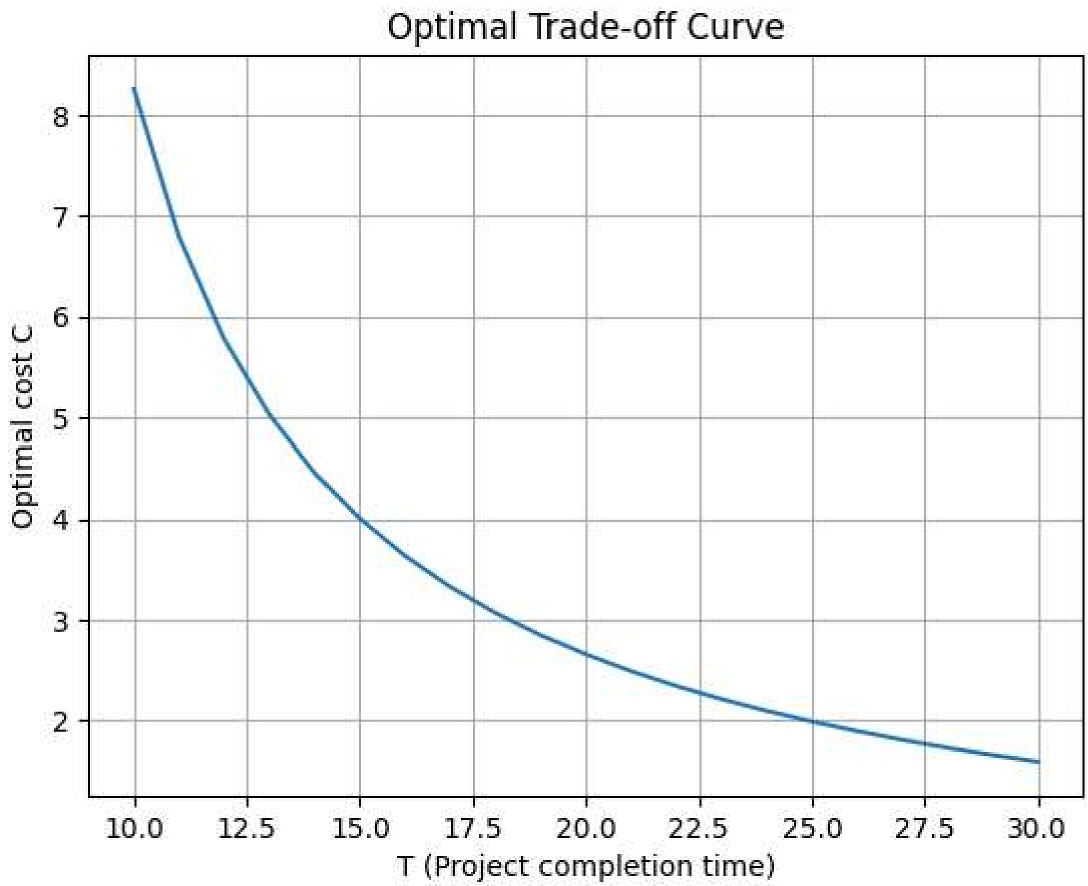
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(1)

# total number of tasks
n = 10

alpha = np.random.rand(n) + 1
m = np.random.rand(n) + 1

P = [
    [0, 6], [1, 2], [1, 3], [1, 4], [1, 6],
    [2, 7], [3, 7], [3, 8], [5, 9], [6, 9], [8, 9]
]


def visualize_schedule(s, f, save=None):
    """
    Visualize a schedule.

    Parameters
    ----------
    s : numpy.ndarray
        Start times of tasks.
    f : numpy.ndarray
        Finish times of tasks.
    save : str, optional
        Path to save the figure.
    """

    fig, ax = plt.subplots(figsize=(8, 4))
    ax.set_position([0.1, 0.1, 0.7, 0.8])

    # compute cost of each task
    cost = [alpha[k] / (f[k] - s[k] - m[k]) for k in range(n)]
    colorbar_max_cost = np.round(1.5 * max(cost), 1)

    # order of display of tasks
    order = np.array([0, 6, 1, 4, 5, 2, 7, 3, 8, 9])

    # plot tasks as rectangles
    for i, k in enumerate(reversed(order)):
        color = plt.cm.BuPu(cost[k] / colorbar_max_cost)
        ax.fill([s[k], f[k], f[k], s[k]], [i, i, i + 1, i + 1], color=color, edgeco
        ax.text((s[k] + f[k]) / 2, i + 0.5, str(k + 1), fontsize=12, ha='center', v

    # plot precedence constraints
    for i, j in P:
        for k in [i, j]:
            ypos = n - np.argwhere(order == k)[0, 0]
            ax.plot([f[i], f[i]], [ypos - 1, ypos], color='red', linewidth=1.5)

    # axes
    T = max(f)
```

```python
    ax.set_xlim([0, T])
    ax.set_xticks(np.arange(0, T + 1, 5))
    ax.set_xlabel('time')
    ax.set_ylim([0, n])
    ax.set_yticks([])
    ax.set_ylabel('task')

    # colorbar
    ax_cbar = fig.add_axes([0.85, 0.1, 0.02, 0.8])
    ax_cbar.imshow(np.linspace(0, 1, 256)[:, None], aspect='auto', origin='lower',
    ax_cbar.set_xticks([])
    ax_cbar.yaxis.tick_right()
    ax_cbar.set_yticks([0, 256])
    ax_cbar.set_yticklabels([0, colorbar_max_cost])
    ax_cbar.set_title('cost', fontsize=10)

    # render
    if save is not None:
        plt.savefig(save, bbox_inches='tight')
    plt.show()
```

In [63]:

```python
eps = 1e-6
s = cp.Variable(n)
f = cp.Variable(n)
T = cp.Parameter()

ts = []
costs = []


for T_val in range(10, 31):

    # Objective: only the convex cost
    objective = cp.Minimize(
        cp.sum(cp.multiply(alpha, cp.inv_pos(f - s - m)))
    )

    constraints = [
        f - s >= m + eps,
        s >= 0,
        f >= 0,
        cp.max(f) <= T_val
    ]

    # precedence constraints
    for (i, j) in P:
        constraints += [s[j] >= f[i]]

    prob = cp.Problem(objective, constraints)
    prob.solve()

    ts.append(T_val)
    costs.append(prob.value)

# Plot
plt.plot(ts, costs)
```

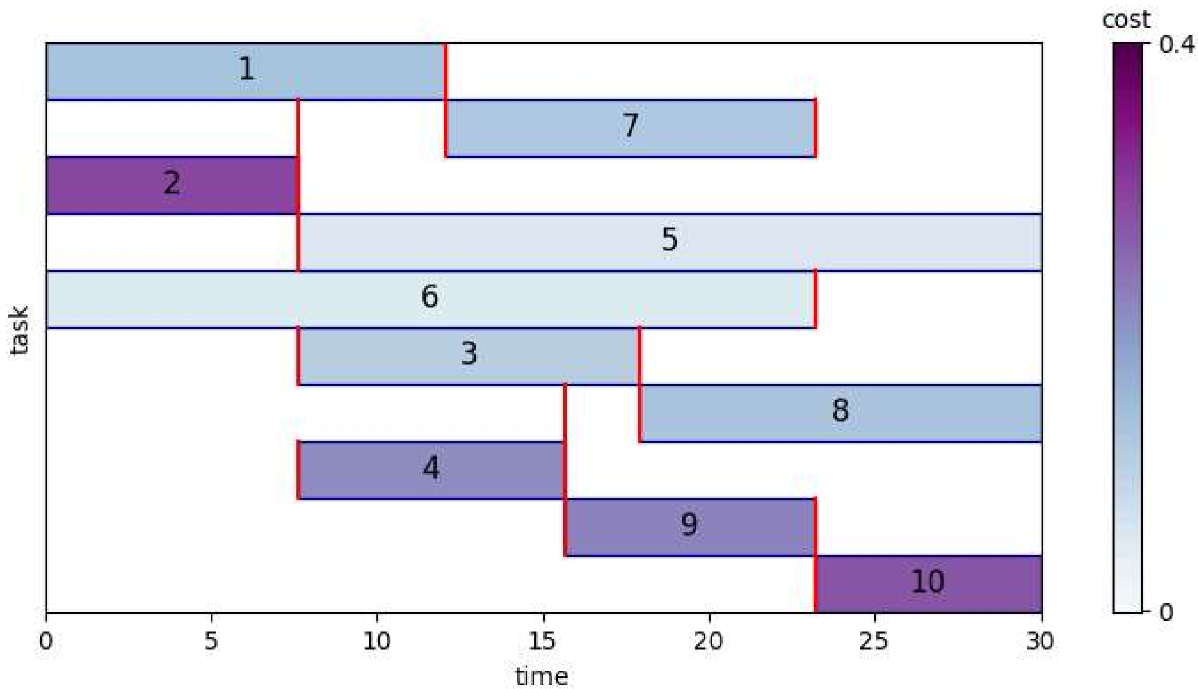
```python
plt.xlabel("T (Project completion time)")
plt.ylabel("Optimal cost C")
plt.title("Optimal Trade-off Curve")
plt.grid(True)
plt.show()
```



In [62]:
```python
objective = cp.Minimize(
    cp.sum(cp.multiply(alpha,cp.inv_pos(f - s - m)))
)

constraints = [
    f - s >= m + eps,
    s >= 0,
    f >= 0,
    cp.max(f) <= 20
]

# precedence constraints
for (i, j) in P:
    constraints += [s[j] >= f[i]]

prob = cp.Problem(objective, constraints)
prob.solve()
```

Out[62]: np.float64(2.656825645230484)

In [65]:
```python
visualize_schedule(s.value, f.value)
```



cost
0.4
0
task
time
0
5
10
15
20
25
30
1
7
2
5
6
3
8
4
9
10