Chapter 1. Getting Started with Pyomo

Pyomo Home Page

- Pyomo home page

Pyomo Book

- Hart, William E., Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola. Pyomo – Optimization Modeling in Python. Second Edition. Vol. 67. Springer, 2017.

Pyomo On-Line Resources

- Read the Docs is the official documentation for the latest release of Pyomo.
- Nicholson, Bethany L., Laird, Carl Damon, Siirola, John Daniel, Watson, Jean-Paul, and Hart, William E. Mon . "Pyomo Tutorial.". United States. https://www.osti.gov/servlets/purl/1376827. Presentation slides in pdf format.

- Pyomo Questions on Stack Overflow
- Pyomo Forum on Google Groups
- Pyomo Examples from the official Pyomo Github Repository.
- PyomoGallery

Pyomo Bibliography

- Hart, William E., Jean-Paul Watson, and David L. Woodruff. "Pyomo: modeling and solving mathematical programs in Python." Mathematical Programming Computation 3(3) (2011): 219-260.
- Nicholson, Bethany, John D. Siirola, Jean-Paul Watson, Victor M. Zavala, and Lorenz T. Biegler. "pyomo.dae: a modeling and automatic discretization framework for optimization with differential and algebraic equations." Mathematical Programming Computation 10(2) (2018): 187-223.
- Watson, Jean-Paul, David L. Woodruff, and William E. Hart. "PySP: modeling and solving stochastic programs in Python." Mathematical Programming Computation 4(2) (2012): 109-149.

In [ ]:

1.1 Installing a Pyomo/Python Development Environment

Step 1. Install Anaconda

Developing scientific and engineering applications in Python requires an interpreter for a particular version of the Python language, a collection of previously developed software libraries, and additional development tools including editors and package managers. Together these elements comprise a Python distribution.

There are many Python distributions available from commercial and free sources. The Anaconda distribution available from Anaconda.com is among the most complete and best known distributions currently available, and is available as a free download or in commercially supported enterprise version. Anaconda includes

- a Python interpreter,
- a user interface Anaconda Navigator providing access to software development tools,
- pre-installed versions of major python libraries,
- the conda package manager to manage python packages and environments.

Installation Procedure:

1. If you have previously installed Anaconda and wish to start over, then a first step is to uninstall the earlier version. While it is possible to maintain multiple versions of Anaconda, there are problems that can arise when installing new packages. Uninstalling prior installations of Anaconda installations is the easiest way to avoid those problems.
2. Download a version of Anaconda appropriate for your laptop. Unless you have a specific reason to use an earlier version of the Python language, download the 64-bit graphical installer for the latest version of Python (currently Python 3.6).
3. Locate and launch the graphical installer from your download directory. Either follow the prompts or consult these more detailed instructions. Normally you will want to use the default choices to install Anaconda into your home folder (a.k.a. directory) for your use only. Generally there is no need to install the optional Microsoft VSCode.
4. Verify that your installation is working. For example, you should be able to locate and lauch a new application Anaconda Navigator.
5. Install any available package updates. Open a command line (either the Terminal application on a Mac located look in the Applications/Utilities folder, or the Command Prompt on Windows), and execute the following two commands on separate lines.

```
conda update conda
conda update anaconda
```

If everything is working correctly, these commands should download and install any recent updates to the Anaconda package.

Step 2. Install Pyomo

The following commands install Pyomo and dependencies. These commands should be executed one at a time from a terminal window on MacOS or a command window on Windows.

```
conda install -c conda-forge pyomo
conda install -c conda-forge pyomo.extras
```

Step 3. Install Solvers

Solvers are needed to compute solutions to the optimization models developed using Pyomo. The solvers glpk (mixed integer linear optimization) and ipopt (nonlinear optimization) cover a wide range of optimization models that arise in process systems engineering and provide good starting point for learning Pyomo. These are installed with the following commands (again, executed one at a time from a terminal window or command prompt).

```
conda install -c conda-forge glpk
conda install -c conda-forge ipopt
```

At this point you should have a working installation of a Python/Pyomo development environment. If you're just getting started with Pyomo, at this point you should be able to use Anaconda Navigator to open a Jupyter session in a browswer, then download and open this notebook in Jupyter. If all's well, the Pyomo model in the cells below should produce useful results.

Step 4. Optional: Compile Ipopt with HSL Solvers

The Ipopt package uses third-party packages for solving linear subproblems which are generally the most time-consuming steps in the solution algorithm. By default, ipopt is distributed with the mumps solvers. If you expect make extensive use of Ipopt for nonlinear problems, or to be solving larger models with 10's of thousands of variables, then you may wish to recompile Ipopt to use high-performance linear solvers available under license from other sources.

In particular, HSL makes available a collection of high-performance solvers for use with Ipopt under a free personal license or a free academic license (commercial licenses are also available). These solvers frequently provide a 10x improvement in solution speed. Instructions for downloading, compiling, and installing the necessary software are available from COIN-OR.

When used with Anaconda, it is convenient to install the recompiled Ipopt directly in the anaconda binary library. Given the instructions above, use the option `--prefix=~anaconda3` when calling `configure`.

Step 5. Optional: Install Additional Solvers

The glpk and ipopt solvers are sufficient to handle meaningful Pyomo models with hundreds to several thousand variables and constraints. These solvers are available under open-source licenses and an excellent starting point for building applications in Pyomo. These may be all you need for many problems. However, as applications get large or more complex, there may be a need to install additional solvers.

Gurobi

Gurobi, for example, is a state-of-the-art high performance commercial solver for large-scale linear, mixed-integer linear, and quadratic programming problems. Unlike glpk, Gurobi is a multi-threaded application taht can take full advantage of a multi-core laptop. Gurobi offers free licenses with one-year terms for academic use, and trial licenses for commercial use. If your application has outgrown glpk, then you'll almost certainly want to give Gurbobi a try.

To install for use in Pyomo, download the standard Gurobi installer and perform the default installation. Follow instructions for registering your copy and obtaining a license. Because Gurobi will be installed outside of the the default Anaconda installation, you will need to specify the actual Gurobi executable. On MacOS, for example, the executable is `/usr/local/bin/gurobi.sh`.

COIN-OR CBC

Solving larger problems may wish to install a multi-threaded solver such as the COIN-OR cbc solvers.

```
conda install -c conda-forge coincbc
```

High performance commercial solvers, such as and CPLEX, are also available at no cost for many academic uses (this is a fantastic deal!), and with trial licenses for commercial use.

In [ ]:

```
```

1.2 Running Pyomo on Google Colab

This note notebook shows how to install the basic pyomo package on Google Colab, and then demonstrates the subsequent installation and use of various solvers including

- GLPK
- COIN-OR CBC
- COIN-OR Ipopt
- COIN-OR Bonmin
- COIN-OR Couenne
- COIN-OR Gecode

Basic Installation of Pyomo

We'll do a quiet installation of pyomo using `pip`. This needs to be done once at the start of each Colab session.

In [1]:

```
!pip install -q pyomo
```

```
    100% |████████████████████████████████| 2.1MB 12.6MB/s
    100% |████████████████████████████████| 51kB 19.4MB/s
    100% |████████████████████████████████| 256kB 28.6MB/s
```

The installation of pyomo can be verified by entering a simple model. We'll use the model again in subsequent cells to demonstrate the installation and execution of various solvers.

In [2]:

```
from pyomo.environ import *

# create a model
model = ConcreteModel()

# declare decision variables
model.x = Var(domain=NonNegativeReals)
model.y = Var(domain=NonNegativeReals)

# declare objective
model.profit = Objective(expr = 40*model.x + 30*model.y, sense=maximize)

# declare constraints
model.demand = Constraint(expr = model.x <= 40)
model.laborA = Constraint(expr = model.x + model.y <= 80)
model.laborB = Constraint(expr = 2*model.x + model.y <= 100)

model.pprint()
```

```
2 Var Declarations
    x : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  None :  None : False :  True : NonNegativeReals
    y : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  None :  None : False :  True : NonNegativeReals

1 Objective Declarations
    profit : Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : maximize : 40*x + 30*y

3 Constraint Declarations
    demand : Size=1, Index=None, Active=True
        Key  : Lower : Body : Upper : Active
        None :  -Inf :    x :  40.0 :   True
    laborA : Size=1, Index=None, Active=True
        Key  : Lower : Body  : Upper : Active
        None :  -Inf : x + y :  80.0 :   True
    laborB : Size=1, Index=None, Active=True
        Key  : Lower : Body    : Upper : Active
        None :  -Inf : 2*x + y : 100.0 :   True

6 Declarations: x y profit demand laborA laborB
```

GLPK Installation

GLPK is a the open-source **G**NU **L**inear **P**rogramming **K**it available for use under the GNU General Public License 3. GLPK is a single-threaded simplex solver generally suited to small to medium scale linear-integer programming problems. It is written in C with minimal dependencies and is therefore highly portable among computers and operating systems. GLPK is often 'good enough' for many examples. For larger problems users should consider higher-performance solvers, such as COIN-OR CBC, that can take advantage of multi-threaded processors.

In [ ]:

```
!apt-get install -y -qq glpk-utils
```

In [5]:

```
SolverFactory('glpk', executable='/usr/bin/glpsol').solve(model).write()

# display solution
print('\nProfit = ', model.profit())

print('\nDecision Variables')
print('x = ', model.x())
print('y = ', model.y())

print('\nConstraints')
print('Demand  = ', model.demand())
print('Labor A = ', model.laborA())
print('Labor B = ', model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: 2600.0
  Upper bound: 2600.0
  Number of objectives: 1
  Number of constraints: 4
  Number of variables: 3
  Number of nonzeros: 6
  Sense: maximize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Termination condition: optimal
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 0
      Number of created subproblems: 0
  Error rc: 0
  Time: 0.020076274871826172
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.0

Decision Variables
x =  20.0
y =  60.0

Constraints
Demand  =  20.0
Labor A =  80.0
Labor B =  100.0
```

COIN-OR CBC Installation

COIN-OR CBC is a multi-threaded open-source **C**oin-or **b**ranch and **c**ut mixed-integer linear programming solver written in C++ under the Eclipse Public License (EPL). CBC is generally a good choice for a general purpose MILP solver for medium to large scale problems.

In [ ]:

```
!apt-get install -y -qq coinor-cbc
```

In [7]:

```
SolverFactory('cbc', executable='/usr/bin/cbc').solve(model).write()

# display solution
print('\nProfit = ', model.profit())

print('\nDecision Variables')
print('x = ', model.x())
print('y = ', model.y())

print('\nConstraints')
print('Demand  = ', model.demand())
print('Labor A = ', model.laborA())
print('Labor B = ', model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: -2600.0
  Upper bound: -2600.0
  Number of objectives: 1
  Number of constraints: 4
  Number of variables: 3
  Number of nonzeros: 6
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  User time: -1.0
  Termination condition: optimal
  Error rc: 0
  Time: 0.019547700881958008
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.0

Decision Variables
x =  20.0
y =  60.0

Constraints
Demand  =  20.0
Labor A =  80.0
Labor B =  100.0
```

COIN-OR Ipopt Installation

COIN-OR Ipopt is an open-source **I**nterior **P**oint **Opt**imizer for large-scale nonlinear optimization available under the Eclipse Public License (EPL). It is well-suited to solving nonlinear programming problems without integer or binary constraints.

In [ ]:

```
!wget -N -q "https://ampl.com/dl/open/ipopt/ipopt-linux64.zip"
!unzip -o -q ipopt-linux64
```

In [9]:

```
SolverFactory('ipopt', executable='/content/ipopt').solve(model).write()

# display solution
print('\nProfit = ', model.profit())

print('\nDecision Variables')
print('x = ', model.x())
print('y = ', model.y())

print('\nConstraints')
print('Demand  = ', model.demand())
print('Labor A = ', model.laborA())
print('Labor B = ', model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 3
  Number of variables: 2
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: Ipopt 3.12.8\x3a Optimal Solution Found
  Termination condition: optimal
  Id: 0
  Error rc: 0
  Time: 0.022800445556640625
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.000025994988

Decision Variables
x =  20.0000001998747
y =  60.0000006

Constraints
Demand  =  20.0000001998747
Labor A =  80.0000007998747
Labor B =  100.0000009997494
```

COIN-OR Bonmin Installation

COIN-OR Bonmin is a **b**asic **o**pen-source solver for **n**onlinear **m**ixed-**in**teger programming problems (MINLP). It utilizes CBC and Ipopt for solving relaxed subproblems.

In [ ]:

```
!wget -N -q "https://ampl.com/dl/open/bonmin/bonmin-linux64.zip"
!unzip -o -q bonmin-linux64
```

In [11]:

```
SolverFactory('bonmin', executable='/content/bonmin').solve(model).write()

# display solution
print('\nProfit = ', model.profit())

print('\nDecision Variables')
print('x = ', model.x())
print('y = ', model.y())

print('\nConstraints')
print('Demand  = ', model.demand())
print('Labor A = ', model.laborA())
print('Labor B = ', model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 0
  Number of variables: 2
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: bonmin\x3a Optimal
  Termination condition: optimal
  Id: 3
  Error rc: 0
  Time: 0.025995254516601562
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.0000259999797

Decision Variables
x =  20.000000199999512
y =  60.00000059999998

Constraints
Demand  =  20.000000199999512
Labor A =  80.0000007999995
Labor B =  100.000000999999
```

COIN-OR Couenne Installation

COIN-OR Couenne](https://www.coin-or.org/Couenne/) is attempts to find global optima for mixed-integer nonlinear programming problems (MINLP).

In [ ]:

```
!wget -N -q "https://ampl.com/dl/open/couenne/couenne-linux64.zip"
!unzip -o -q couenne-linux64
```

In [13]:

```
SolverFactory('couenne', executable='/content/couenne').solve(model).write()

# display solution
print('\nProfit = ', model.profit())

print('\nDecision Variables')
print('x = ', model.x())
print('y = ', model.y())

print('\nConstraints')
print('Demand  = ', model.demand())
print('Labor A = ', model.laborA())
print('Labor B = ', model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 0
  Number of variables: 2
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: couenne\x3a Optimal
  Termination condition: optimal
  Id: 3
  Error rc: 0
  Time: 0.023235797882080078
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.00002599998

Decision Variables
x =  20.000000199999512
y =  60.00000059999999

Constraints
Demand  =  20.000000199999512
Labor A =  80.0000007999995
Labor B =  100.00000099999902
```

Gecode Installation

In [ ]:

```
!wget -N -q "https://ampl.com/dl/open/gecode/gecode-linux64.zip"
!unzip -o -q gecode-linux64
```

Gecode solves constraint programming problems and does not support continuous variables. We therefore create a second model using exclusively discrete variables.

In [15]:

```
from pyomo.environ import *

# create a model
discrete_model = ConcreteModel()

# declare decision variables
discrete_model.x = Var(domain=NonNegativeIntegers)
discrete_model.y = Var(domain=NonNegativeIntegers)

# declare objective
discrete_model.profit = Objective(expr = 40*discrete_model.x + 30*discrete_model.y, sen
se=maximize)

# declare constraints
discrete_model.demand = Constraint(expr = discrete_model.x <= 40)
discrete_model.laborA = Constraint(expr = discrete_model.x + discrete_model.y <= 80)
discrete_model.laborB = Constraint(expr = 2*discrete_model.x + discrete_model.y <= 100)

discrete_model.pprint()
```

```
2 Var Declarations
    x : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  None :  None : False :  True : NonNegativeIntegers
    y : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  None :  None : False :  True : NonNegativeIntegers

1 Objective Declarations
    profit : Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : maximize : 40*x + 30*y

3 Constraint Declarations
    demand : Size=1, Index=None, Active=True
        Key  : Lower : Body : Upper : Active
        None :  -Inf :    x :  40.0 :   True
    laborA : Size=1, Index=None, Active=True
        Key  : Lower : Body  : Upper : Active
        None :  -Inf : x + y :  80.0 :   True
    laborB : Size=1, Index=None, Active=True
        Key  : Lower : Body    : Upper : Active
        None :  -Inf : 2*x + y : 100.0 :   True

6 Declarations: x y profit demand laborA laborB
```

In [16]:

```
SolverFactory('gecode', executable='/content/gecode').solve(discrete_model).write()

# display solution
print('\nProfit = ', discrete_model.profit())

print('\nDecision Variables')
print('x = ', discrete_model.x())
print('y = ', discrete_model.y())

print('\nConstraints')
print('Demand  = ', discrete_model.demand())
print('Labor A = ', discrete_model.laborA())
print('Labor B = ', discrete_model.laborB())
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 0
  Number of variables: 2
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: gecode 4.4.0\x3a optimal solution; 201 nodes, 0 fails, objective 2600
  Termination condition: optimal
  Id: 0
  Error rc: 0
  Time: 0.019869565963745117
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0

Profit =  2600.0

Decision Variables
x =  20.0
y =  60.0

Constraints
Demand  =  20.0
Labor A =  80.0
Labor B =  100.0
```

In [ ]:

1.3 Running Pyomo on the Notre Dame CRC Cluster

Preliminaries

Before proceeding with this tutorial, you need to obtain a CRC account and install a private copy of Pyomo on the CRC cluster.

Request a CRC Account

First, you must register for a CRC account. These are free for all ND researchers.

Install Pyomo on CRC Cluster

Per CRC policies, standalone Python packages such as Pyomo are not centrally installed or maintained. Instead, users must install a private copy of standalone Python packages on the CRC cluster. This can be easily done in a few steps:

First, `ssh` into a CRC interactive node. For example,

```
ssh crcfe02.crc.nd.edu
```

Next (optional), may load your preferred version of Python. If you do not, the system default (2.7) will be used. Pyomo supports both 2.7 and 3.x:

```
module load python/3.6.4
```

Then, install Pyomo using `pip`, a popular package manager for Python:

```
pip install --user pyomo
```

By default, Pyomo will be installed in `.local/bin`. See the ND CRC wiki instructions for adding `.local/bin` to your PATH.

Running on the CRC

There are two ways to run Pyomo on the CRC cluster: on an interactive mode (for testing only) or by submitting a job to the queue. The following shows how to do this for a simple optimization problem.

Test Problem

We will consider the following simple linear program to test Pyomo on the CRC:

```
# Load Pyomo
from pyomo.environ import *

# Create model
m = ConcreteModel()
m.x = Var([1,2,3], domain=NonNegativeReals)
m.c1 = Constraint(expr = m.x[1] + m.x[2] + m.x[3] <= 1)
m.c2 = Constraint(expr = m.x[1] + 2*m.x[2] >= 0.3)
m.OBJ = Objective(expr = m.x[3], sense=maximize)

# Specify solver
solver=SolverFactory('ipopt')

# Solve model
solver.solve(m, tee=True)
```

Copy this code and save it on your CRC AFS space as `pyomo_test.py`.

Running From the Command Line

The easest option is to directly run this code on an interactive node on the CRC cluster.

1. Login to the CRC cluster:

```
ssh crcfe02.crc.nd.edu
```

1. Load your preferred version of python and your favorite optimization solver.

```
module load ipopt
```

1. Run the Python script

```
python pyomo_test.py
```

You should see the following output:

```
Ipopt 3.12.8:

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

This is Ipopt version 3.12.8, running with linear solver ma27.

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        5
Number of nonzeros in Lagrangian Hessian.............:        0

Total number of variables............................:        3
                     variables with only lower bounds:        3
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints                .:        0
```

```
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        2
        inequality constraints with only lower bounds:        1
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        1

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 -9.9999900e-03 2.70e-01 6.00e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1 -2.3096613e-02 1.48e-01 3.42e+00  -1.7 1.87e-01    -  1.10e-01 4.70e-01h  1
   2 -7.1865937e-02 0.00e+00 8.39e-01  -1.7 1.42e-01    -  4.87e-01 1.00e+00f  1
   3 -8.1191843e-01 0.00e+00 4.70e-01  -1.7 1.47e+00    -  1.00e+00 5.02e-01f  1
   4 -7.9286713e-01 0.00e+00 2.00e-07  -1.7 1.91e-02    -  1.00e+00 1.00e+00f  1
   5 -8.4402296e-01 0.00e+00 3.48e-03  -3.8 5.61e-02    -  9.40e-01 9.12e-01f  1
   6 -8.4949514e-01 0.00e+00 1.50e-09  -3.8 7.04e-03    -  1.00e+00 1.00e+00f  1
   7 -8.4999437e-01 0.00e+00 1.84e-11  -5.7 4.99e-04    -  1.00e+00 1.00e+00f  1
   8 -8.5000001e-01 0.00e+00 2.51e-14  -8.6 5.65e-06    -  1.00e+00 1.00e+00f  1

Number of Iterations....: 8

                                   (scaled)                 (unscaled)
Objective...............:  -8.5000001246787615e-01   -8.5000001246787615e-01
Dual infeasibility......:   2.5143903682766679e-14    2.5143903682766679e-14
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   2.5342284749259295e-09    2.5342284749259295e-09
Overall NLP error.......:   2.5342284749259295e-09    2.5342284749259295e-09


Number of objective function evaluations             = 9
Number of objective gradient evaluations             = 9
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 9
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 9
Number of Lagrangian Hessian evaluations             = 8
Total CPU secs in IPOPT (w/o function evaluations)   =      0.004
Total CPU secs in NLP function evaluations           =      0.000

EXIT: Optimal Solution Found.
```

Submitting to a Queue

Per CRC policies, the interactive nodes should only be used for testing short computational jobs. All other jobs should be submitted to a queue.

To submit the previous project on the CRC queue, write a script that contains the following:

```
#!/bin/csh

#$ -M netid@nd.edu     # Email address for job notification
#$ -m abe                    # Send mail when job aborts, begins, and ends
#$ -q long                  # Specify queue
#$ -N job_name      # Specify job name

module load python/3.6.4     # Required modules
module load ipopt

python3 pyomo_test.py        # Command to execute
```

Name it something logical, such as `submission_script`. Then type:

```
qsub submission_script
```

You should get an email when your job begins/ends/aborts. You can also monitor its process through

```
qstat –u username
```

More info is available at the ND CRC wiki Quick Start Guide.

Available Solvers

The CRC clusters supports several large-scale optimization solvers that are directly callable from Pyomo. This makes it increadily easy - often one only needs to change a few lines of code - to try different solvers for an optimization problem.

The following contains instructions on using solvers currently available on the CRC cluster. By modifying the `SolverFactory` in `pyomo_test.py`, you can try all of these solvers on the simple test problem.

CPLEX

First load the module:

```
module load cplex
```

This will update your environmental variables and the Gurobi executable should be callable from Pyomo:

```
solver=SolverFactory('cplex')
```

Here is the output for the test problem:

```
Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.7.1.0
  with Simplex, Mixed Integer & Barrier Optimizers
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
Copyright IBM Corp. 1988, 2017.  All Rights Reserved.

Type 'help' for a list of available commands.
Type 'help' followed by a command name for more
information on commands.

CPLEX> Logfile 'cplex.log' closed.
Logfile '/afs/crc.nd.edu/user/a/adowling/Private/tmp103f3a.cplex.log' open.
CPLEX> Problem '/afs/crc.nd.edu/user/a/adowling/Private/tmpdhMJXY.pyomo.lp' read
.
Read time = 0.01 sec. (0.00 ticks)
CPLEX> Problem name         :
/afs/crc.nd.edu/user/a/adowling/Private/tmpdhMJXY.pyomo.lp
Objective sense      : Maximize
Variables            :       4
Objective nonzeros   :       1
Linear constraints   :       3  [Less: 1,  Greater: 1,  Equal: 1]
  Nonzeros           :       6
  RHS nonzeros       :       3

Variables            : Min LB: 0.000000         Max UB: all infinite
Objective nonzeros   : Min   : 1.000000         Max   : 1.000000
Linear constraints   :
  Nonzeros           : Min   : 1.000000         Max   : 2.000000
  RHS nonzeros       : Min   : 0.3000000        Max   : 1.000000
CPLEX> CPXPARAM_Read_APIEncoding                         "UTF-8"
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 2 rows, 3 columns, and 5 nonzeros.
Presolve time = 0.00 sec. (0.00 ticks)
Initializing dual steep norms . . .

Iteration log . . .
Iteration:     1   Dual objective     =             0.850000

Dual simplex - Optimal:  Objective =  8.5000000000e-01
Solution time =    0.00 sec.  Iterations = 1 (0)
Deterministic time = 0.00 ticks  (4.51 ticks/sec)

CPLEX> Solution written to file
'/afs/crc.nd.edu/user/a/adowling/Private/tmp0ryhXG.cplex.sol'.
```

Gurobi

First load the module:

```
module load gurobi
```

This will update your environmental variables and the Gurobi executable should be callable from Pyomo:

```
solver=SolverFactory('gurobi')
```

Here is the output for the test problem:

```
Optimize a model with 3 rows, 4 columns and 6 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [3e-01, 1e+00]
Presolve removed 1 rows and 1 columns
Presolve time: 0.02s
Presolved: 2 rows, 3 columns, 5 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.0000000e+00   1.500000e-01   0.000000e+00      0s
       1    8.5000000e-01   0.000000e+00   0.000000e+00      0s

Solved in 1 iterations and 0.02 seconds
Optimal objective  8.500000000e-01
Freed default Gurobi environment
```

Ipopt

Two versions (modules) of Ipopt are available to CRC users: `ipopt/3.12.8` and `ipopt/hsl/3.12.8`. The latter supports HSL linear algebra routines whereas the former does not (and relies on the open-source library MUMPS). In general, the HSL linear algebra routines are more stable and are significantly faster for large-scale problems. All IPOPT users are strongly encouraged to use the HSL libraries. To obtain access to `ipopt/hsl/3.12.8`, ND users must apply for a free HSL academic license and then forward the approval email to CRCSupport@nd.edu.

To use either version of Ipopt, first load the module. For example,

```
module load ipopt/hsl/3.12.8
```

This will update your environmental variables and the ipopt executable should be callable from Pyomo:

```
solver=SolverFactory('ipopt')
```

If needed, you can also directly specify the path to the Ipopt executable to Pyomo:

```
solver=SolverFactory('ipopt',
executable="/afs/crc.nd.edu/x86_64_linux/i/ipopt/3.12.8-hsl/bin/ipopt")
```

Next, you may want to customize some of the Ipopt options. For example, specify the linear algebra routine (use MA57 from the HSL library):

```
solver.options['linear_solver'] = "ma57"
```

Here is the output when using MA57:

```
Ipopt 3.12.8: linear_solver=ma57


******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

This is Ipopt version 3.12.8, running with linear solver ma57.

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        5
Number of nonzeros in Lagrangian Hessian.............:        0

Total number of variables............................:        3
                     variables with only lower bounds:        3
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        2
        inequality constraints with only lower bounds:        1
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        1

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 -9.9999900e-03 2.70e-01 6.00e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1 -2.3096613e-02 1.48e-01 3.42e+00  -1.7 1.87e-01    -  1.10e-01 4.70e-01h  1
   2 -7.1865937e-02 0.00e+00 8.39e-01  -1.7 1.42e-01    -  4.87e-01 1.00e+00f  1
   3 -8.1191843e-01 0.00e+00 4.70e-01  -1.7 1.47e+00    -  1.00e+00 5.02e-01f  1
   4 -7.9286713e-01 0.00e+00 2.00e-07  -1.7 1.91e-02    -  1.00e+00 1.00e+00f  1
   5 -8.4402296e-01 0.00e+00 3.48e-03  -3.8 5.61e-02    -  9.40e-01 9.12e-01f  1
   6 -8.4949514e-01 0.00e+00 1.50e-09  -3.8 7.04e-03    -  1.00e+00 1.00e+00f  1
   7 -8.4999437e-01 0.00e+00 1.84e-11  -5.7 4.99e-04    -  1.00e+00 1.00e+00f  1
   8 -8.5000001e-01 0.00e+00 2.51e-14  -8.6 5.65e-06    -  1.00e+00 1.00e+00f  1

Number of Iterations....: 8

                                   (scaled)                 (unscaled)
Objective...............:  -8.5000001246787615e-01   -8.5000001246787615e-01
Dual infeasibility......:   2.5143903682766679e-14    2.5143903682766679e-14
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   2.5342284749259295e-09    2.5342284749259295e-09
Overall NLP error.......:   2.5342284749259295e-09    2.5342284749259295e-09


Number of objective function evaluations             = 9
Number of objective gradient evaluations             = 9
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 9
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 9
Number of Lagrangian Hessian evaluations             = 8
Total CPU secs in IPOPT (w/o function evaluations)   =      0.004
Total CPU secs in NLP function evaluations           =      0.000

EXIT: Optimal Solution Found.
```

Coin-OR

Several solvers are availabe in the COIN-OR module.

The first step to using any of these solvers is loading the module:

```
module load coin-or
```

Clp

The Clp solver executable should now be callable from Pyomo:

```
solver=SolverFactory('clp')
```

If needed, you can explicitly specify the path to the executable:

```
solver=SolverFactory('clp', executable="/afs/crc.nd.edu/x86_64_linux/c/coin-or/clp/1.16.11/bin/clp")
```

Output for the test problem:

```
Coin LP version 1.16.11, build Jun  4 2018
command line - /afs/crc.nd.edu/x86_64_linux/c/coin-or/clp/1.16.11/bin/clp
/afs/crc.nd.edu/user/a/adowling/Private/tmpEsGwQH.pyomo.nl -AMPL
```

Cbc

The Cbc solver executable should now be callable from Pyomo:

```
solver=SolverFactory('cbc')
```

If needed, you can explicitly specify the path to the executable:

```
solver=SolverFactory('cbc', executable="/afs/crc.nd.edu/x86_64_linux/c/coin-or/c
bc/2.9.9/bin/cbc")
```

Output for the test problem:

```
Welcome to the CBC MILP Solver
Version: 2.9.6
Build Date: Jun  3 2018

command line - /afs/crc.nd.edu/x86_64_linux/c/coin-or/cbc/2.9.9/bin/cbc -
printingOptions all -import
/afs/crc.nd.edu/user/a/adowling/Private/tmp5jMyQq.pyomo.lp -stat=1 -solve -solu
/afs/crc.nd.edu/user/a/adowling/Private/tmp5jMyQq.pyomo.soln (default strategy
1)
Option for printingOptions changed from normal to all
 CoinLpIO::readLp(): Maximization problem reformulated as minimization
Presolve 2 (-1) rows, 3 (-1) columns and 5 (-1) elements
Statistics for presolved model


Problem has 2 rows, 3 columns (1 with objective) and 5 elements
There are 1 singletons with objective
Column breakdown:
3 of type 0.0->inf, 0 of type 0.0->up, 0 of type lo->inf,
0 of type lo->up, 0 of type free, 0 of type fixed,
0 of type -inf->0.0, 0 of type -inf->up, 0 of type 0.0->1.0
Row breakdown:
0 of type E 0.0, 1 of type E 1.0, 0 of type E -1.0,
0 of type E other, 0 of type G 0.0, 0 of type G 1.0,
1 of type G other, 0 of type L 0.0, 0 of type L 1.0,
0 of type L other, 0 of type Range 0.0->1.0, 0 of type Range other,
0 of type Free
Presolve 2 (-1) rows, 3 (-1) columns and 5 (-1) elements
0  Obj 0 Primal inf 1.299998 (2) Dual inf 0.999999 (1)
2  Obj -0.85
Optimal - objective value -0.85
After Postsolve, objective -0.85, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective -0.85 - 2 iterations time 0.002, Presolve 0.00
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.01
```

Bonmin

Warning: Bonmin has been compiled using the HSL linear algebra routines. All users that have access to Ipopt with HSL also have access to Bonmin.

After you submit your HSL license approval email to CRCsupport@nd.edu, the Bonmin solver executable will be callable from Pyomo:

```
solver=SolverFactory('bonmin')
```

If needed, you can explicitly specify the path to the executable:

```
solver=SolverFactory('bonmin', executable="/afs/crc.nd.edu/x86_64_linux/c/coin-o
r/bonmin/1.8.6/bin/bonmin")
```

Output for the test problem:

```
Bonmin 1.8.6 using Cbc 2.9.9 and Ipopt 3.12.8
bonmin:
Cbc3007W No integer variables - nothing to do

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

NLP0012I
              Num      Status      Obj             It       time
ocation
NLP0014I             1         OPT -0.85000002        6 0.00295
Cbc3007W No integer variables - nothing to do

    "Finished"
```

Couenne

Warning: Couenne has been compiled using the HSL linear algebra routines. All users that have access to Ipopt with HSL also have access to Couenne.

After you submit your HSL license approval email to CRCsupport@nd.edu, the Couenne solver executable will be callable from Pyomo:

```
solver=SolverFactory('couenne')
```

If needed, you can explicitly specify the path to the executable:

```
solver=SolverFactory('couenne', executable="/afs/crc.nd.edu/x86_64_linux/c/coin-
or/couenne/0.5.6/bin/couenne")
```

Output for the test problem:

```
Couenne 0.5.6 -- an Open-Source solver for Mixed Integer Nonlinear Optimization
Mailing list: couenne@list.coin-or.org
Instructions: http://www.coin-or.org/Couenne
couenne:
ANALYSIS TEST: Couenne: new cutoff value -8.5000000000e-01 (0.045811 seconds)
NLP0012I
              Num      Status      Obj             It       time
ocation
NLP0014I             1         OPT -0.85000002        9 0.004882
Loaded instance "/afs/crc.nd.edu/user/a/adowling/Private/tmps3yXIf.pyomo.nl"
Constraints:            2
Variables:              3 (0 integer)
Auxiliaries:            1 (0 integer)

Coin0506I Presolve 2 (-1) rows, 3 (-1) columns and 5 (-2) elements
Clp0006I 0  Obj -0.849815 Primal inf 0.00018408442 (1) Dual inf 0.999999 (1)
Clp0006I 2  Obj -0.85
Clp0000I Optimal - objective value -0.85
Clp0032I Optimal objective -0.85 - 2 iterations time 0.002, Presolve 0.00
Cbc3007W No integer variables - nothing to do
Clp0000I Optimal - objective value -0.85

    "Finished"

Linearization cuts added at root node:          3
Linearization cuts added in total:              3  (separation time: 6.5e-05s)
Total solve time:                        0.001362s (0.001362s in branch-and-boun
)
Lower bound:                                 -0.85
Upper bound:                                 -0.85  (gap: 0.00%)
Branch-and-bound nodes:                          0
Performance of                            FBBT:      2.6e-05s,        3 runs. fix
0 shrnk:    10.7635 ubd:   0.666667 2ubd:          0 infeas:          0
```

SCIP

Coming soon.

In [ ]:

```
```

Chapter 2. Linear Programming

2.1 Production Models with Linear Constraints

This notebook demonstrates the use of linear programming to maximize profit for a simple model of a multiproduct production facility. The notebook uses Pyomo to represent the model with the glpk solver to calculate solutions.

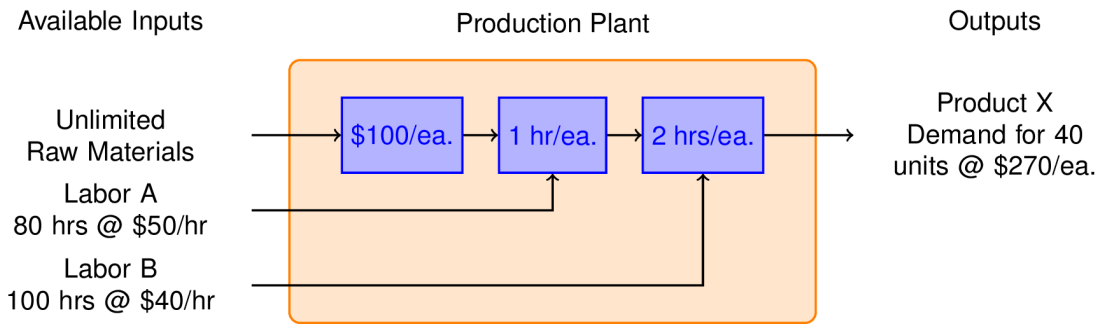
In [ ]:

```
%%capture
!pip install -q pyomo
!apt-get install -y -qq coinor-cbc
```

Example: Production Plan for a Single Product Plant

Suppose you are thinking about starting up a business to produce Product X. You have determined there is a market for X of up to 40 units per week at a price of USD 270 each. The production of each unit requires USD 100 of raw materials, 1 hour of type A labor, and 2 hours of type B labor. You have an unlimited amount of raw material available to you, but only 80 hours per week of labor A at a cost of USD 50/hour, and 100 hours per week of labor B at a cost of USD 40 per hour. Ignoring all other expenses, what is the maximum weekly profit?

To get started on this problem, we sketch a flow diagram illustrating the flow of raw materials and labor through the production plant.



The essential decision we need to make is how many units or Product X to produce each week. That's our *decision variable* which we denote as x. The weekly revenues are then

$$\text{Revenue} = \$270x$$

The costs include the value of the raw materials and each form of labor. If we produce x units a week, then the total cost is

$$\text{Cost} = \underbrace{\$100x}_{\text{Raw Material}} + \underbrace{\$50x}_{\text{Labor A}} + \underbrace{2 \times \$40x}_{\text{Labor B}} = \$230x$$

We see immediately that the gross profit is just

$$\begin{aligned} \text{Profit} &= \text{Revenue} - \text{Cost} \\ &= \$270x - \$230x \\ &= \$40x \end{aligned}$$

which means there is a profit earned on each unit of X produced, so let's produce as many as possible.

There are three constraints that limit how many units can be produced. There is market demand for no more than 40 units per week. Producing $x = 40$ units per week will require 40 hours per week of Labor A, and 80 hours per week of Labor B. Checking those constraints we see that we have enough labor of each type, so the maximum profit will be

$$\max \text{Profit} = \$$$

What we conclude is that market demand is the 'most constraining constraint.' Once we've made that deduction, the rest is a straightforward problem that can be solved by inspection.

Pyomo Model

While this problem can be solved by inspection, here we show a Pyomo model that generates a solution to the problem.

In [2]:

```
from pyomo.environ import *
model = ConcreteModel()

# declare decision variables
model.x = Var(domain=NonNegativeReals)

# declare objective
model.profit = Objective(
    expr = 40*model.x,
    sense = maximize)

# declare constraints
model.demand = Constraint(expr = model.x <= 40)
model.laborA = Constraint(expr = model.x <= 80)
model.laborB = Constraint(expr = 2*model.x <= 100)

# solve
SolverFactory('cbc').solve(model).write()
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: -1600.0
  Upper bound: -1600.0
  Number of objectives: 1
  Number of constraints: 4
  Number of variables: 2
  Number of nonzeros: 4
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  User time: -1.0
  Termination condition: optimal
  Error rc: 0
  Time: 0.01619696617126465
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

The results of the solution step show the solver has converged to an optimal solution. Next we display the particular components of the model of interest to us.

In [3]:

```
print("Profit = ", model.profit(), " per week")
print("X = ", model.x(), " units per week")
```

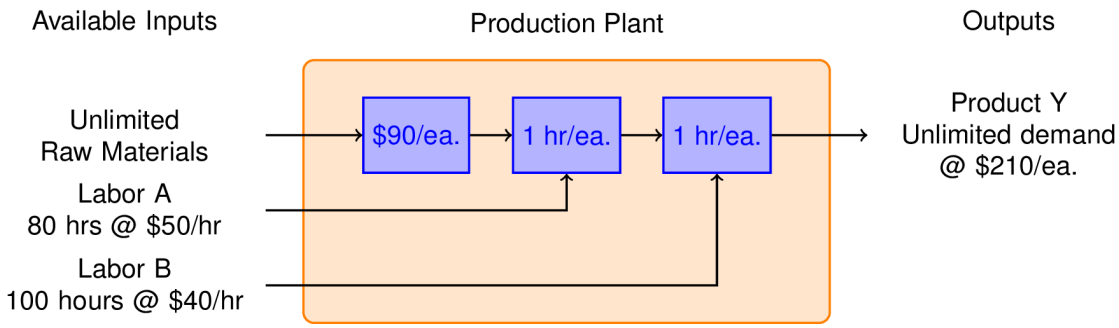
```
Profit =  1600.0  per week
X =  40.0  units per week
```

Exercises

1. Suppose the demand could be increased to 50 units per month. What would be the increased profits? What if the demand increased to 60 units per month? How much would you be willing to pay for your marketing department for the increased demand?
2. Increase the cost of LaborB. At what point is it no longer financially viable to run the plant?

Production Plan: Product Y

Your marketing department has developed plans for a new product called Y. The product sells at a price of USD 210/each, and they expect that you can sell all that you can make. It's also cheaper to make, requiring only USD 90 in raw materials, 1 hour of Labor type A at USD 50 per hour, and 1 hour of Labor B at USD 40 per hour. What is the potential weekly profit?



In [4]:

```
from pyomo.environ import *
model = ConcreteModel()

# declare decision variables
model.y = Var(domain=NonNegativeReals)

# declare objective
model.profit = Objective(
    expr = 30*model.y,
    sense = maximize)

# declare constraints
model.laborA = Constraint(expr = model.y <= 80)
model.laborB = Constraint(expr = model.y <= 100)

# solve
SolverFactory('cbc').solve(model).write()
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: -2400.0
  Upper bound: -2400.0
  Number of objectives: 1
  Number of constraints: 3
  Number of variables: 2
  Number of nonzeros: 3
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  User time: -1.0
  Termination condition: optimal
  Error rc: 0
  Time: 0.022749662399291992
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

In [5]:

```
print("Profit = ", model.profit())
print("Units of Y = ", model.y())
```

```
Profit =  2400.0
Units of Y =  80.0
```

Compared to product X, we can manufacture and sell up 80 units per week for a total profit of $2,400. This is very welcome news.

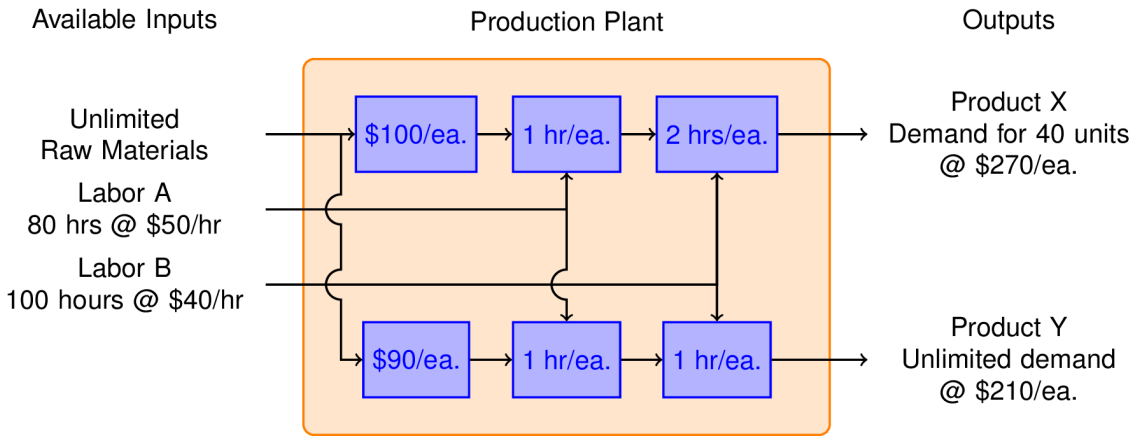
Exercises

1. What is the limiting resource? That is, which of the two types of labor limits the capacity of your plant to produce more units of Y?
2. What rate would you be willing to pay for the additional labor necessary to increase the production of Y?

Production Plan: Mixed Product Strategy

So far we have learned that we can make $1,600 per week by manufacturing product X, and $2,400 per week manufacturing product Y. Is it possible to do even better?

To answer this question, we consider the possibilty of manufacturing both products in the same plant. The marketing department assures us that product Y will not affect the sales of product X. So the same constraints hold as before, but now we have two decision variables, x and y.



In [6]:

```
from pyomo.environ import *
model = ConcreteModel()

# declare decision variables
model.x = Var(domain=NonNegativeReals)
model.y = Var(domain=NonNegativeReals)

# declare objective
model.profit = Objective(
    expr = 40*model.x + 30*model.y,
    sense = maximize)

# declare constraints
model.demand = Constraint(expr = model.x <= 40)
model.laborA = Constraint(expr = model.x + model.y <= 80)
model.laborB = Constraint(expr = 2*model.x + model.y <= 100)

# solve
SolverFactory('cbc').solve(model).write()
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: -2600.0
  Upper bound: -2600.0
  Number of objectives: 1
  Number of constraints: 4
  Number of variables: 3
  Number of nonzeros: 6
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  User time: -1.0
  Termination condition: optimal
  Error rc: 0
  Time: 0.02334761619567871
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

In [7]:

```
# display solution
print("Profit = ", model.profit())
print("Units of X = ", model.x())
print("Units of Y = ", model.y())
```

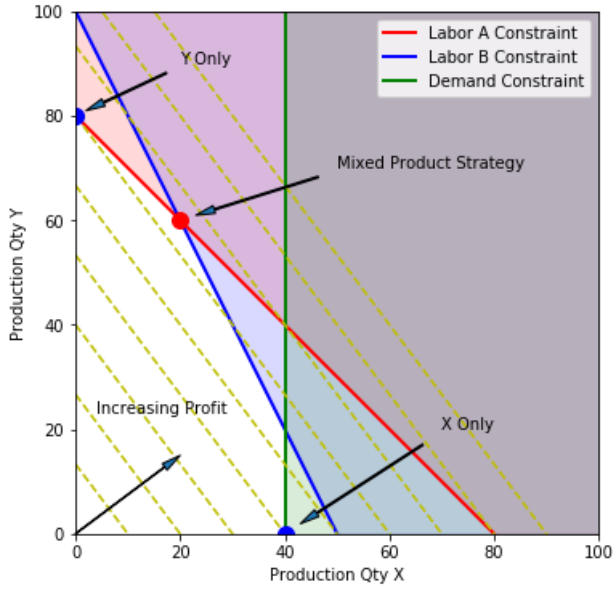
```
Profit =  2600.0
Units of X =  20.0
Units of Y =  60.0
```

The mixed product strategy earns more profit than either of the single product srategies. Does this surprise you? Before going further, try to explain why it is possible for a mixed product strategy to earn more profit than either of the possible single product strategies.

What are the active constraints?

In [8]:

```
%matplotlib inline
from pylab import *

figure(figsize=(6,6))
subplot(111, aspect='equal')
axis([0,100,0,100])
xlabel('Production Qty X')
ylabel('Production Qty Y')

# Labor A constraint
x = array([0,80])
y = 80 - x
plot(x,y,'r',lw=2)
fill_between([0,80,100],[80,0,0],[100,100,100],color='r',alpha=0.15)

# Labor B constraint
x = array([0,50])
y = 100 - 2*x
plot(x,y,'b',lw=2)
fill_between([0,50,100],[100,0,0],[100,100,100],color='b',alpha=0.15)

# Demand constraint
plot([40,40],[0,100],'g',lw=2)
fill_between([40,100],[0,0],[100,100],color='g',alpha=0.15)

legend(['Labor A Constraint','Labor B Constraint','Demand Constraint'])

# Contours of constant profit
x = array([0,100])
for p in linspace(0,3600,10):
    y = (p - 40*x)/30
    plot(x,y,'y--')

# Optimum
plot(20,60,'r.',ms=20)
annotate('Mixed Product Strategy', xy=(20,60), xytext=(50,70),
         arrowprops=dict(shrink=.1,width=1,headwidth=5))

plot(0,80,'b.',ms=20)
annotate('Y Only', xy=(0,80), xytext=(20,90),
         arrowprops=dict(shrink=0.1,width=1,headwidth=5))

plot(40,0,'b.',ms=20)
annotate('X Only', xy=(40,0), xytext=(70,20),
         arrowprops=dict(shrink=0.1,width=1,headwidth=5))

text(4,23,'Increasing Profit')
annotate('', xy=(20,15), xytext=(0,0),
         arrowprops=dict(width=0.5,headwidth=5))

savefig('LPprob01.png',bbox_inches='tight')
```



What is the incremental value of labor?

In [9]:

```
from pyomo.environ import *
model = ConcreteModel()

# for access to dual solution for constraints
model.dual = Suffix(direction=Suffix.IMPORT)

# declare decision variables
model.x = Var(domain=NonNegativeReals)
model.y = Var(domain=NonNegativeReals)

# declare objective
model.profit = Objective(
    expr = 40*model.x + 30*model.y,
    sense = maximize)

# declare constraints
model.demand = Constraint(expr = model.x <= 40)
model.laborA = Constraint(expr = model.x + model.y <= 80)
model.laborB = Constraint(expr = 2*model.x + model.y <= 100)

# solve
SolverFactory('cbc').solve(model).write()
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: -2600.0
  Upper bound: -2600.0
  Number of objectives: 1
  Number of constraints: 4
  Number of variables: 3
  Number of nonzeros: 6
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  User time: -1.0
  Termination condition: optimal
  Error rc: 0
  Time: 0.023309707641601562
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

Analysis of the constraints.

In [10]:

```
str = "   {0:7.2f} {1:7.2f} {2:7.2f} {3:7.2f}"

print("Constraint  value  lslack  uslack    dual")
for c in [model.demand,model.laborA,model.laborB]:
    print(c, str.format(c(), c.lslack(), c.uslack(), model.dual[c]))
```

```
Constraint  value  lslack  uslack    dual
demand      20.00     inf   20.00    0.00
laborA      80.00     inf    0.00  -20.00
laborB     100.00     inf    0.00  -10.00
```

Theory of Constraints

- For n decisions you should expect to find n 'active' constraints.
- Each inactive constraint has an associated 'slack.' The associated resources have no incremental value.
- Each active constraint has an associated 'shadow price'. This is additional value of additional resources.

In [ ]:

```

```

2.2 Linear Blending Problem

In [ ]:

```
%%capture
!pip install -q pyomo
!apt-get install -y -qq coinor-cbc
```

Problem Statement (Jenchura, 2017)

A brewery receives an order for 100 gallons of 4% ABV (alchohol by volume) beer. The brewery has on hand beer A that is 4.5% ABV that cost USD 0.32 per gallon to make, and beer B that is 3.7% ABV and cost USD 0.25 per gallon. Water could also be used as a blending agent at a cost of USD 0.05 per gallon. Find the minimum cost blend that meets the customer requirements.

Representing Problem Data as a Python Dictionary

We will use this problem as an opportunity to write a Python function that accepts data on raw materials and customer specifications to produce the lowest cost blend.

The first step is to represent the problem data in a generic manner that could, if needed, be extended to include additional blending components. Here we use a dictionary of materials, each key denoting a blending agent. For each key there is a sub-dictionary containing attributes of each blending component.

In [ ]:

```
data = {
    'A': {'abv': 0.045, 'cost': 0.32},
    'B': {'abv': 0.037, 'cost': 0.25},
    'W': {'abv': 0.000, 'cost': 0.05},
}
```

Model Formulation

Objective Function

If we let subscript c denote a blending component from the set of blending components C, and denote the volume of c used in the blend as x_c, the cost of the blend is

$$\text{cost} = \sum_{c \in C} x_c P_c$$

where P_c is the price per unit volume of c. Using the Python data dictionary defined above, the price P_c is given by `data[c]['cost']`.

Volume Constraint

The customer requirement is produce a total volume V. Assuming ideal solutions, the constraint is given by

$$V = \sum_{c \in C} x_c$$

where x_c denotes the volume of component c used in the blend.

Product Composition Constraint

The product composition is specified as 4% alchohol by volume. Denoting this as $\bar{A}$, the constraint may be written as

$$\bar{A} = \frac{\sum_{c \in C} x_c A_c}{\sum_{c \in C} x_c}$$

where A_c is the alcohol by volume for component c. As written, this is a nonlinear constraint. Multiplying both sides of the equation by the denominator yields a linear constraint

$$\bar{A} \sum_{c \in C} x_c = \sum_{c \in C} x_c A_c$$

A final form for this constraint can be given in either of two versions. In the first version we subtract the left-hand side from the right to give

$$0 = \sum_{c \in C} x_c \left(A_c - \bar{A} \right) \qquad \text{Version 1 of the linear blending constraint}$$

Alternatively, the summation on the left-hand side corresponds to total volume. Since that is known as part of the problem specification, the blending constraint could also be written as

$$\bar{A} V = \sum_{c \in C} x_c A_c \qquad \text{Version 2 of the linear blending constraint}$$

Which should you use? Either will generally work well. The advantage of version 1 is that it is fully specified by a product requirement $\bar{A}$, which is sometimes helpful in writing elegant Python code.

Implementation in Pyomo

A Pyomo implementation of this blending model is shown in the next cell. The model is contained within a Python function so that it can be more easily reused for additional calculations, or eventually for use by the process operator.

Note that the pyomo library has been imported with the prefix `pyomo`. This is good programming practive to avoid namespace collisions with problem data.

In [4]:

```
import pyomo.environ as pyomo

vol = 100
abv = 0.040

def beer_blend(vol, abv, data):
    C = data.keys()
    model = pyomo.ConcreteModel()
    model.x = pyomo.Var(C, domain=pyomo.NonNegativeReals)
    model.cost = pyomo.Objective(expr = sum(model.x[c]*data[c]['cost'] for c in C))
    model.vol = pyomo.Constraint(expr = vol == sum(model.x[c] for c in C))
    model.abv = pyomo.Constraint(expr = 0 == sum(model.x[c]*(data[c]['abv'] - abv) for c
in C))

    solver = pyomo.SolverFactory('cbc')
    solver.solve(model)

    print('Optimal Blend')
    for c in data.keys():
        print('   ', c, ':', model.x[c](), 'gallons')
    print()
    print('Volume = ', model.vol(), 'gallons')
    print('Cost = $', model.cost())

beer_blend(vol, abv, data)
```

```
Optimal Blend
   A : 37.5 gallons
   B : 62.5 gallons
   W : 0.0 gallons

Volume =  100.0 gallons
Cost = $ 27.625
```

In [ ]:

2.3 Design of a Cold Weather Fuel for a Camping Stove

Problem Statement

The venerable alcohol stove has been invaluable camping accessory for generations. They are simple, reliable, and in a pinch, can be made from aluminum soda cans.

Alcohol stoves are typically fueled with denatured alcohol. Denatured alcohol, sometimes called methylated spirits, is a generally a mixture of ethanol and other alcohols and compounds designed to make it unfit for human consumption. An MSDS description of one manufacturer's product describes a roughly fifity/fifty mixture of ethanol and methanol.

The problem with alcohol stoves is they can be difficult to light in below freezing weather. The purpose of this notebook is to design of an alternative cold weather fuel that could be mixed from other materials commonly available from hardware or home improvement stores.

In [ ]:

```
!pip install -q pyomo
!apt-get install -y -qq coinor-cbc
```

Vapor Pressure Data

The following data was collected for potential fuels commonly available at hardware and home improvement stores. The data consists of price ($/gal.) and parameters to predict vapor pressure using the Antoine equation,

$$\log_{10} P_s^{vap}(T) = A_s - \frac{B_s}{T + C_s}$$

where the subscript s refers to species, temperature T is in units of degrees Celcius, and pressure P is in units of mmHg. The additional information for molecular weight and specific gravity will be needed to present the final results in volume fraction.

In [1]:

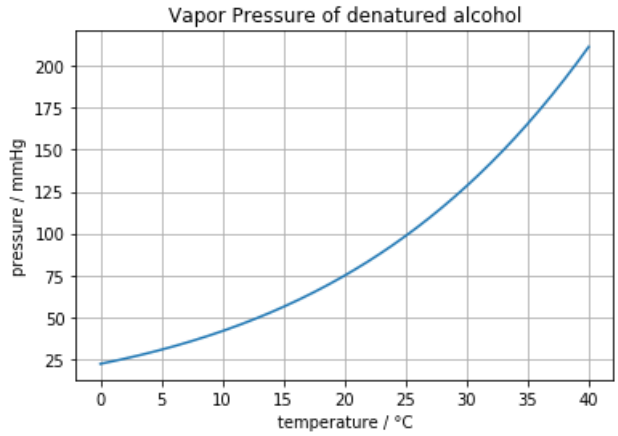
```
data = {
    'ethanol'          : {'MW':  46.07, 'SG': 0.791, 'A': 8.04494, 'B': 1554.3,  'C': 2
2.65},
    'methanol'         : {'MW':  32.04, 'SG': 0.791, 'A': 7.89750, 'B': 1474.08, 'C': 2
9.13},
    'isopropyl alcohol': {'MW':  60.10, 'SG': 0.785, 'A': 8.11778, 'B': 1580.92, 'C': 2
19.61},
    'acetone'          : {'MW':  58.08, 'SG': 0.787, 'A': 7.02447, 'B': 1161.0,  'C': 2
4.0},
    'xylene'           : {'MW': 106.16, 'SG': 0.870, 'A': 6.99052, 'B': 1453.43, 'C': 21
5.31},
    'toluene'          : {'MW':  92.14, 'SG': 0.865, 'A': 6.95464, 'B': 1344.8,  'C': 2
9.48},
}
```

The first step is to determine the vapor pressure of denatured alcohol over a typical range of operating temperatures. For this we assume denatured alcohol is a 40/60 (mole fraction) mixture of ethanol and methanol.

In [2]:

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

def Pvap(T, s):
    return 10**(data[s]['A'] - data[s]['B']/(T + data[s]['C']))

def Pvap_denatured(T):
    return 0.4*Pvap(T, 'ethanol') + 0.6*Pvap(T, 'methanol')

T = np.linspace(0, 40, 200)

plt.plot(T, Pvap_denatured(T))
plt.title('Vapor Pressure of denatured alcohol')
plt.xlabel('temperature / °C')
plt.ylabel('pressure / mmHg')
print("Vapor Pressure at 0C =", round(Pvap_denatured(0),1), "mmHg")
plt.grid(True)
```

```
Vapor Pressure at 0C = 22.1 mmHg
```



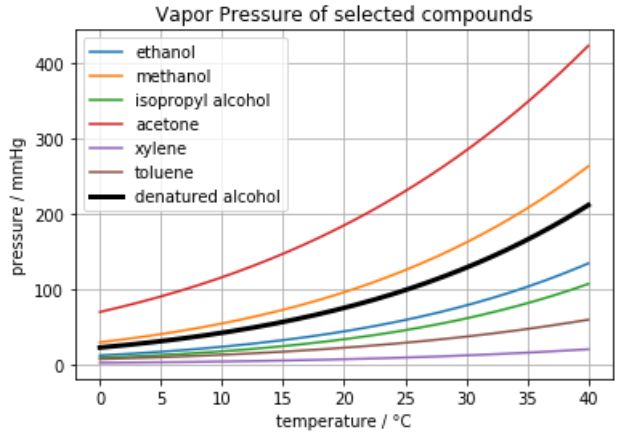
Cold Weather Product Requirements

We seek a cold weather fuel with increased vapor pressure at 0°C and lower, and also provides safe and normal operation of the alcohol stove at higher operating temperatures.

For this purpose, we seek a mixture of commonly available liquids with a vapor pressure of at least 22 mmHg at the lowest possible temperature, and no greater than the vapor pressure of denatured alcohol at temperatures 30°C and above.

In [4]:

```
for s in data.keys():
    plt.plot(T, Pvap(T,s))
plt.plot(T, Pvap_denatured(T), 'k', lw=3)
plt.legend(list(data.keys()) + ['denatured alcohol'])
plt.title('Vapor Pressure of selected compounds')
plt.xlabel('temperature / °C')
plt.ylabel('pressure / mmHg')
plt.grid(True)
```



Optimization Model

The first optimization model is to create a mixture that maximizes the vapor pressure at -10°C while having a vapor pressure less than or equal to denatured alcohol at 30°C and above.

The decision variables in the optimization model correspond to x_s, the mole fraction of each species $s \in S$ from the set of available species S. By definition, the mole fractions must satisfy

$$x_s \geq 0 \qquad \forall s \in S$$
$$\sum_{s \in S} x_s = 1$$

The objective is to maximize the vapor pressure at low temperatures, say -10°C, while maintaing a vapor pressure less than or equal to denatured alcohol at 30°C. Using Raoult's law for ideal mixtures,

$$\max_{x_s} \sum_{s \in S} x_s P_s^{vap}(-10 \quad ^\circ \quad C)$$

subject to

$$\sum_{s \in S} x_s P_s^{vap}(30 \quad \leq P_{denatured\ alcohol}^{vap} C)(30 \quad ^\circ \quad C)$$

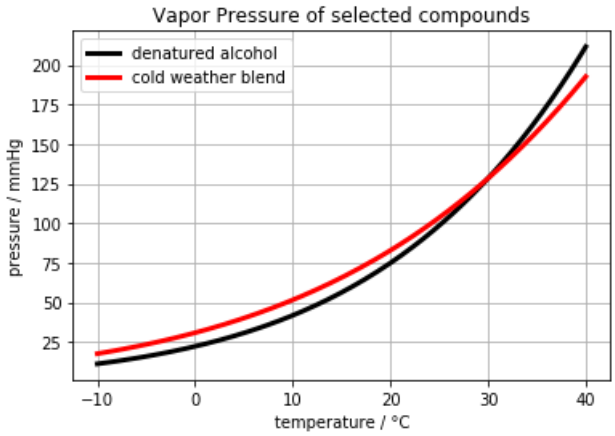
Pyomo Implementation and Solution

This optimization model is implemented in Pyomo in the following cell.

In [5]:

```
import pyomo.environ as pyomo

m = pyomo.ConcreteModel()

S = data.keys()
m.x = pyomo.Var(S, domain=pyomo.NonNegativeReals)

def Pmix(T):
    return sum(m.x[s]*Pvap(T,s) for s in S)

m.obj = pyomo.Objective(expr = Pmix(-10), sense=pyomo.maximize)

m.cons = pyomo.ConstraintList()

m.cons.add(sum(m.x[s] for s in S)==1)
m.cons.add(Pmix(30) <= Pvap_denatured(30))
m.cons.add(Pmix(40) <= Pvap_denatured(40))

solver = pyomo.SolverFactory('cbc')
solver.solve(m)

print("Vapor Pressure at -10°C =", m.obj(), "mmHg")

T = np.linspace(-10,40,200)
plt.plot(T, Pvap_denatured(T), 'k', lw=3)
plt.plot(T, [Pmix(T)() for T in T], 'r', lw=3)
plt.legend(['denatured alcohol'] + ['cold weather blend'])
plt.title('Vapor Pressure of selected compounds')
plt.xlabel('temperature / °C')
plt.ylabel('pressure / mmHg')
plt.grid(True)
```

```
Vapor Pressure at -10°C = 17.48178543436185 mmHg
```



The Pandas library is useful for summarizing the solution in tabular form.

In [6]:

```
import pandas as pd

s = data.keys()
results = pd.DataFrame.from_dict(data).T
for s in S:
    results.loc[s,'mole fraction'] = m.x[s]()

MW = sum(m.x[s]()*data[s]['MW'] for s in S)
for s in S:
    results.loc[s,'mass fraction'] = m.x[s]()*data[s]['MW']/MW

vol = sum(m.x[s]()*data[s]['MW']/data[s]['SG'] for s in S)
for s in S:
    results.loc[s,'vol fraction'] = m.x[s]()*data[s]['MW']/data[s]['SG']/vol

results
```

Out[6]:

	A	B	C	MW	SG	mole fraction	mass fraction	vol fraction
ethanol	8.04494	1554.30	222.65	46.07	0.791	0.000000	0.0000	0.000000
methanol	7.89750	1474.08	229.13	32.04	0.791	0.000000	0.0000	0.000000
isopropyl alcohol	8.11778	1580.92	219.61	60.10	0.785	0.000000	0.0000	0.000000
acetone	7.02447	1161.00	224.00	58.08	0.787	0.428164	0.2906	0.311695
xylene	6.99052	1453.43	215.31	106.16	0.870	0.571836	0.7094	0.688305
toluene	6.95464	1344.80	219.48	92.14	0.865	0.000000	0.0000	0.000000

In [ ]:

```
```

2.4 Gasoline Blending

In [ ]:

```
%%capture
!pip install -q pyomo
!apt-get install -y -qq coinor-cbc
```

In [ ]:

```
import pandas as pd
import pyomo.environ as pyomo
```

Gasoline Product Specifications

The gasoline products include regular and premium gasoline. In addition to the current price, the specifications include

- **octane** the minimum road octane number. Road octane is the computed as the average of the Research Octane Number (RON) and Motor Octane Number (MON).
- **Reid Vapor Pressure** Upper and lower limits are specified for the Reid vapor pressure. The Reid vapor pressure is the absolute pressure exerted by the liquid at 100°F.
- **benzene** the maximum volume percentage of benzene allowed in the final product. Benzene helps to increase octane rating, but is also a treacherous environmental contaminant.

In [3]:

```
products = {
    'Regular' : {'price': 2.75, 'octane': 87, 'RVPmin': 0.0, 'RVPmax': 15.0, 'benzene':
1.1},
    'Premium' : {'price': 2.85, 'octane': 91, 'RVPmin': 0.0, 'RVPmax': 15.0, 'benzene':
1.1},
}

print(pd.DataFrame.from_dict(products).T)
```

```
         RVPmax  RVPmin  benzene  octane  price
Regular    15.0     0.0      1.1    87.0   2.75
Premium    15.0     0.0      1.1    91.0   2.85
```

Stream Specifications

A typical refinery produces many intermediate streams that can be incorporated in a blended gasoline product. Here we provide data on seven streams that include:

- **Butane** n-butane is a C4 product stream produced from the light components of the crude being processed by the refinery. Butane is a highly volatile of gasoline.
- **LSR** Light straight run naptha is a 90°F to 190°F cut from the crude distillation column primarily consisting of straight chain C5-C6 hydrocarbons.
- **Isomerate** is the result of isomerizing LSR to produce branched molecules that results in higher octane number.
- **Reformate** is result of catalytic reforming heavy straight run napthenes to produce a high octane blending component, as well by-product hydrogen used elsewhere in the refinery for hydro-treating.
- **Reformate LB** is a is a low benzene variant of reformate.
- **FCC Naphta** is the product of a fluidized catalytic cracking unit designed to produce gasoline blending components from long chain hydrocarbons present in the crude oil being processed by the refinery.
- **Alkylate** The alkylation unit reacts iso-butane with low-molecular weight alkenes to produce a high octane blending component for gasoline.

The stream specifications include research octane and motor octane numbers for each blending component, the Reid vapor pressure, the benzene content, cost, and availability (in gallons per day). The road octane number is computed as the average of the RON and MON.

In [4]:

```
streams = {
    'Butane'       : {'RON': 93.0, 'MON': 92.0, 'RVP': 54.0, 'benzene': 0.00, 'cost':
0.85, 'avail': 30000},
    'LSR'          : {'RON': 78.0, 'MON': 76.0, 'RVP': 11.2, 'benzene': 0.73, 'cost': 2
05, 'avail': 35000},
    'Isomerate'    : {'RON': 83.0, 'MON': 81.1, 'RVP': 13.5, 'benzene': 0.00, 'cost': 2
.20, 'avail': 0},
    'Reformate'    : {'RON':100.0, 'MON': 88.2, 'RVP':  3.2, 'benzene': 1.85, 'cost':
2.80, 'avail': 60000},
    'Reformate LB' : {'RON': 93.7, 'MON': 84.0, 'RVP':  2.8, 'benzene': 0.12, 'cost': 2
.75, 'avail': 0},
    'FCC Naphtha'  : {'RON': 92.1, 'MON': 77.1, 'RVP':  1.4, 'benzene': 1.06, 'cost': 2
.60, 'avail': 70000},
    'Alkylate'     : {'RON': 97.3, 'MON': 95.9, 'RVP':  4.6, 'benzene': 0.00, 'cost':
2.75, 'avail': 40000},
}

# calculate road octane as (R+M)/2
for s in streams.keys():
    streams[s]['octane'] = (streams[s]['RON'] + streams[s]['MON'])/2

# display feed information
print(pd.DataFrame.from_dict(streams).T)
```

```
               MON    RON   RVP    avail  benzene  cost  octane
Butane        92.0   93.0  54.0  30000.0     0.00  0.85   92.50
LSR           76.0   78.0  11.2  35000.0     0.73  2.05   77.00
Isomerate     81.1   83.0  13.5      0.0     0.00  2.20   82.05
Reformate     88.2  100.0   3.2  60000.0     1.85  2.80   94.10
Reformate LB  84.0   93.7   2.8      0.0     0.12  2.75   88.85
FCC Naphtha   77.1   92.1   1.4  70000.0     1.06  2.60   84.60
Alkylate      95.9   97.3   4.6  40000.0     0.00  2.75   96.60
```

Blending Model

This simplified blending model assumes the product attributes can be computed as linear volume weighted averages of the component properties. Let the decision variable $x_{s,p} \geq 0$ be the volume, in gallons, of blending component $s \in S$ used in the final product $p \in P$.

The objective is maximize profit, which is the difference between product revenue and stream costs.

$$\text{profit} = \max_{x_{s,p}} \left(\sum_{p \in P} \text{Price}_p \sum_{s \in S} x_{s,p} - \sum_{s \in S} \text{Cost}_s \sum_{p \in P} x_{s,p} \right)$$

or

$$\text{profit} = \max_{x_{s,p}} \left(\sum_{p \in P} \sum_{s \in S} x_{s,p} \text{Price}_p - \sum_{p \in P} \sum_{s \in S} x_{s,p} \text{Cost}_s \right)$$

The blending constraint for octane can be written as

$$\frac{\sum_{s \in S} x_{s,p} \text{Octane}_s}{\sum_{s \in S} x_{s,p}} \geq \text{Octane}_p \qquad \forall p \in P$$

where Octane_s refers to the octane rating of stream s, whereas Octane_p refers to the octane rating of product p. Multiplying through by the denominator, and consolidating terms gives

$$\sum_{s \in S} x_{s,p} \left(\text{Octane}_s - \text{Octane}_p \right) \geq 0 \qquad \forall p \in P$$

The same assumptions and development apply to the benzene constraint

$$\sum_{s \in S} x_{s,p} \left(\text{Benzene}_s - \text{Benzene}_p \right) \leq 0 \qquad \forall p \in P$$

Reid vapor pressure, however, follows a somewhat different mixing rule. For the Reid vapor pressure we have

$$\sum_{s \in S} x_{s,p} \left(\text{RVP}_s^{1.25} - \text{RVP}_{min,p}^{1.25} \right) \geq 0 \qquad \forall p \in P$$

$$\sum_{s \in S} x_{s,p} \left(\text{RVP}_s^{1.25} - \text{RVP}_{max,p}^{1.25} \right) \leq 0 \qquad \forall p \in P$$

Pyomo Implementation

This model is implemented in the following cell.

In [5]:

```
import pyomo.environ as pyomo

# create model
m = pyomo.ConcreteModel()

# create decision variables
S = streams.keys()
P = products.keys()
m.x = pyomo.Var(S,P, domain=pyomo.NonNegativeReals)

# objective
revenue = sum(sum(m.x[s,p]*products[p]['price'] for s in S) for p in P)
cost = sum(sum(m.x[s,p]*streams[s]['cost'] for s in S) for p in P)
m.profit = pyomo.Objective(expr = revenue - cost, sense=pyomo.maximize)

# constraints
m.cons = pyomo.ConstraintList()
for s in S:
    m.cons.add(sum(m.x[s,p] for p in P) <= streams[s]['avail'])
for p in P:
    m.cons.add(sum(m.x[s,p]*(streams[s]['octane'] -    products[p]['octane'])       for
s in S) >= 0)
    m.cons.add(sum(m.x[s,p]*(streams[s]['RVP']**1.25 - products[p]['RVPmin']**1.25) for
s in S) >= 0)
    m.cons.add(sum(m.x[s,p]*(streams[s]['RVP']**1.25 - products[p]['RVPmax']**1.25) for
s in S) <= 0)
    m.cons.add(sum(m.x[s,p]*(streams[s]['benzene'] -   products[p]['benzene'])      for
s in S) <= 0)

# solve
solver = pyomo.SolverFactory('cbc')
solver.solve(m)

# display results
vol = sum(m.x[s,p]() for s in S for p in P)
print("Total Volume =", round(vol, 1), "gallons.")
print("Total Profit =", round(m.profit(), 1), "dollars.")
print("Profit =", round(100*m.profit()/vol,1), "cents per gallon.")
```

```
Total Volume = 235000.0 gallons.
Total Profit = 100425.0 dollars.
Profit = 42.7 cents per gallon.
```

Displaying the Solution

By Refinery Stream

In [6]:

```
stream_results = pd.DataFrame()
for s in S:
    for p in P:
        stream_results.loc[s,p] = round(m.x[s,p](), 1)
    stream_results.loc[s,'Total'] = round(sum(m.x[s,p]() for p in P), 1)
    stream_results.loc[s,'Available'] = streams[s]['avail']

stream_results['Unused (Slack)'] = stream_results['Available'] - stream_results['Total'
]
print(stream_results)
```

```
              Regular  Premium    Total  Available  Unused (Slack)
Butane        21754.6   8245.4  30000.0    30000.0             0.0
LSR            9211.6  25788.4  35000.0    35000.0             0.0
Isomerate         0.0      0.0      0.0        0.0             0.0
Reformate     19783.9  40216.1  60000.0    60000.0             0.0
Reformate LB      0.0      0.0      0.0        0.0             0.0
FCC Naphtha   70000.0      0.0  70000.0    70000.0             0.0
Alkylate         -0.0  40000.0  40000.0    40000.0             0.0
```

By Refinery Product

In [7]:

```
product_results = pd.DataFrame()
for p in P:
    product_results.loc[p,'Volume'] = round(sum(m.x[s,p]() for s in S), 1)
    product_results.loc[p,'octane'] = round(sum(m.x[s,p]()*streams[s]['octane'] for s in
S)
                                            /product_results.loc[p,'Volume'], 1)
    product_results.loc[p,'RVP'] = round((sum(m.x[s,p]()*streams[s]['RVP']**1.25 for s i
n S)
                                            /product_results.loc[p,'Volume'])**0.8, 1)
    product_results.loc[p,'benzene'] = round(sum(m.x[s,p]()*streams[s]['benzene'] for s
in S)
                                            /product_results.loc[p,'Volume'], 1)
print(product_results)
```

```
           Volume  octane   RVP  benzene
Regular  120750.0    87.0  15.0      1.0
Premium  114250.0    91.0  10.6      0.8
```

Exercises

Expand product list with mid-grade gasoline

The marketing team says there is an opportunity to create a mid-grade gasoline product with a road octane number of 89 that would sell for $2.82/gallon, and with all other specifications the same. Could an additional profit be created?

Create a new cell (or cells) below to compute a solution to this exercise.

In [ ]:

```

```

Impact of regulatory change

New environmental regulations have reduced the allowable benzene levels from 1.1 vol% to 0.62 vol%, and the maximum Reid vapor pressure from 15.0 to 9.0.

Compared to the base case (i.e., without the midgrade product), how does this change profitability?

In [ ]:

```
```

Impact of a change in refinery operations

Given the new product specifications in Exercise 2, let's consider using different refinery streams. In place of Reformate, the refinery could produce Reformate LB. (That is, one or the other of the two streams could be 60000 gallons per day, but not both). Same for LSR and Isomerate. How should the refinery be operated to maximize profitability?

In [ ]:

```
```

2.5 Model Predictive Control of a Double Integrator

In [0]:

```
%%capture
!pip install -q pyomo
!apt-get install -y -qq coinor-cbc
```

Model

The double integrator model is a canonical second order linear system often used to demostrate control principles. A typical example is Newton's second law where a frictionless mass m is subject to external forces in one dimension

$$m\frac{d^2x}{dt^2} = f(t)$$

where x is position and $f(t)$ is the applied force. It is also reasonably approximates the response of a motor to torque inputs, to a ball moving on a beam that can be tilted, and other mechanical systems. Here we consider a case where the control input $f(t)$ and the position $x(t)$ are both bounded in magnitude.

$$\begin{aligned} |\ f(t) &\le F\ | \\ |\ x(t) &\le L\ | \end{aligned}$$

Introducing scalling rules

$$y = \frac{x}{L} \qquad u = \frac{f}{F} \qquad \tau = \frac{t}{T}$$

results in the equation

$$\frac{mL}{T^2F}\frac{d^2y}{d\tau^2} = u$$

Choosing the time scale T as

$$T = \sqrt{\frac{mL}{F}}$$

reduces the control problem to a dimensionless form

$$\frac{d^2y}{d\tau^2} = u$$

subject to constraints

$$\begin{aligned} |\ u(\tau) &|\ \ \forall \tau \in [0,1] \\ |\ y(\tau) &1\ |\ \ \forall \tau \in [0,1] \end{aligned}$$

A variety of control problems that can be formulated from this simple model. Here we consider the problem of determining the range of possible initial conditions $y(0)$ and $\dot{y}(0)$ that can be steered back to a steady position at the origin (i.e., $y = 0$ and $\dot{y} = 0$) without violating the constraints on position or applied control action. For the ball-on-beam experiment, this would correspond to finding intial conditions for the position and velocity of the ball that can be steered back to a steady position at the center of the beam without falling off in the meanwhile.

Discrete Time Approximation

In order to directly construct an optimization model, here we will consider a discrete-time approximation to the double integrator model. We will assume values of $u(\tau)$ are fixed at discrete points in time $\tau_k = kh$ where $k = 0, 1, \ldots, N$ and $h = \frac{T}{N}$ is the sampling time. The control input is held constant between these sample points.

Using the notation $x_1 = y$ and $x_2 = \dot{y}$ we have

$$\begin{aligned} \frac{dx_1}{d\tau} &= x_2 \\ \frac{dx_2}{d\tau} &= u \end{aligned}$$

Because u is constant between sample instants, integrating the second equation gives

$$x_2(\tau_k + h) = x_2(\tau_k) + hu(\tau_k)$$

Substituting and integrating the first equation then yields the pair of equations

$$\begin{aligned} x_1(\tau_k + h) &= x_1(\tau_k) + hx_2(\tau_k) + \frac{h^2}{2}u(\tau_k) \\ x_2(\tau_k + h) &= x_2(\tau_k) + hu(\tau_k) \end{aligned}$$

This discretization gives

$$\begin{aligned} \underbrace{\begin{bmatrix} x_1(\tau_k + h) \\ x_2(\tau_k + h) \end{bmatrix}}_{x(\tau_{k+1})} &= \underbrace{\begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1(\tau_k) \\ x_2(\tau_k) \end{bmatrix}}_{x(\tau_k)} + \underbrace{\begin{bmatrix} \frac{h^2}{2} \\ h \end{bmatrix}}_{B} u(\tau_k) \\ y(\tau_k) &= \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} x_1(\tau_k) \\ x_2(\tau_k) \end{bmatrix}}_{x(\tau_k)} \end{aligned}$$

where $y(\tau_k)$ corresponds to position. The constraints are

$$\begin{aligned} | \quad u(\tau_k) \quad | &\quad \forall\, k = 0, 1, \ldots, N \\ | \quad y(\tau_k) \quad | &\quad \forall\, k = 0, 1, \ldots, N \end{aligned}$$

For the purposes here, we will neglect constaints on the dynamics during the periods between sample points. Any issues with intersample dynamics can be addressed by increasing the number of sample points.

Model Predictive Control

Given values of the state variables $x_1(\tau_0)$ and $x_2(\tau_0)$ and sampling time $h = \frac{T}{N}$ the computational task is to find a control policy $u(\tau_k), u(\tau_{k+1}), \ldots, u(\tau_{k+N-1})$ that steers the state to the origin at t_{k+N}. The model equations are

$$\begin{aligned} x_1(\tau_{k+1}) &= x_1(\tau_k) + h x_2(\tau_k) + \frac{h^2}{2} u(\tau_k) \\ x_2(\tau_{k+1}) &= x_2(\tau_k) + h u(\tau_k) \\ y(\tau_k) &= x_1(\tau_k) \end{aligned}$$

for $k = 0, 1, \ldots, N-1$, subject to final conditions

$$\begin{aligned} x_1(\tau_{k+N}) &= 0 \\ x_2(\tau_{k+N}) &= 0 \end{aligned}$$

and path constraints

$$\begin{array}{|c|c|} u(\tau_k) & \forall k = 0, 1, 2, \ldots, N-1 \\ y(\tau_k) & \forall k = 0, 1, 2, \ldots, N-1 \end{array}$$

The path constraints need to be recast for the purposes of linear optimization. Here we introduce additional decision variables

$$\begin{aligned} u(\tau_k) &= u^+(\tau_k) - u^-(\tau_k) \\ y(\tau_k) &= y^+(\tau_k) - y^-(\tau_k) \end{aligned}$$

where

$$\begin{aligned} 0 \leq u^+(\tau_k), u^-(\tau_k) \leq 1 \\ 0 \leq y^+(\tau_k), y^-(\tau_k) \leq 1 \end{aligned}$$

The objective function is then to minimize

$$\min \sum_{k=0}^{N} \gamma \left[u^+(\tau_k) + u^-(\tau_k) \right] + (1 - \gamma) \left[y^+(\tau_k) + y^-(\tau_k) \right]$$

for a choice of $0 < \gamma < 1$ that represents a desired tradeoff between path constraints on $u(\tau_k)$ and $y(\tau_k)$.

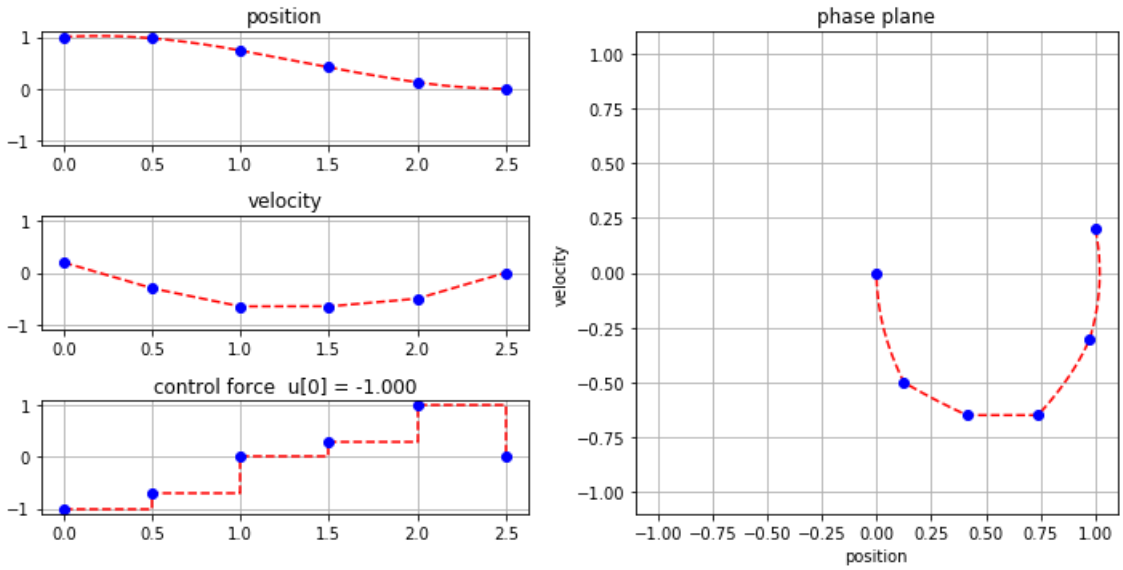
In [0]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import math
import numpy as np
from pyomo.environ import *

def mpc_double_integrator(N=2, h=1):
    m = ConcreteModel()
    m.states = RangeSet(1, 2)
    m.k = RangeSet(0, N)

    m.h = Param(initialize=h, mutable=True)
    m.ic = Param(m.states, initialize={1:0.5, 2:0.5}, mutable=True)
    m.gamma = Param(default=0.5, mutable=True)

    m.x = Var(m.states, m.k)
    m.icfix = Constraint(m.states, rule = lambda m, i: m.x[i,0] == m.ic[i])
    m.x[1,N].fix(0)
    m.x[2,N].fix(0)

    m.u = Var(m.k, bounds=(-1, 1))
    m.upos = Var(m.k, bounds=(0, 1))
    m.uneg = Var(m.k, bounds=(0, 1))
    m.usum = Constraint(m.k, rule = lambda m, k: m.u[k] == m.upos[k] - m.uneg[k])

    m.y = Var(m.k, bounds=(-1, 1))
    m.ypos = Var(m.k, bounds=(0, 1))
    m.yneg = Var(m.k, bounds=(0, 1))
    m.ysum = Constraint(m.k, rule = lambda m, k: m.y[k] == m.ypos[k] - m.yneg[k])

    m.x1_update = Constraint(m.k, rule = lambda m, k:
           m.x[1,k+1] == m.x[1,k] + m.h*m.x[2,k] + m.h**2*m.u[k]/2 if k < N else
Constraint.Skip)
    m.x2_update = Constraint(m.k, rule = lambda m, k:
           m.x[2,k+1] == m.x[2,k] + m.h*m.u[k] if k < N else Constraint.Skip)
    m.y_output = Constraint(m.k, rule = lambda m, k: m.y[k] == m.x[1,k])

    m.uobj = m.gamma*sum(m.upos[k] + m.uneg[k] for k in m.k)
    m.yobj = (1-m.gamma)*sum(m.ypos[k] + m.yneg[k] for k in m.k)
    m.obj = Objective(expr = m.uobj + m.yobj, sense=minimize)

    return m
```

Visualization

In [0]:

```
from itertools import chain

def plot_results(m):
    results = SolverFactory('cbc').solve(m)
    if str(results.solver.termination_condition) != "optimal":
        print(results.solver.termination_condition)
        return

    # solution data at sample times
    h = m.h()
    K = np.array([k for k in m.k])
    u = [m.u[k]() for k in K]
    y = [m.y[k]() for k in K]
    v = [m.x[2,k]() for k in K]

    # interpolate between sample times
    t = np.linspace(0, h)
    tp = [_ for _ in chain.from_iterable(k*h + t for k in K[:-1])]
    up = [_ for _ in chain.from_iterable(u[k] + t*0 for k in K[:-1])]
    yp = [_ for _ in chain.from_iterable(y[k] + t*(v[k] + t*u[k]/2) for k in K[:-1])]
    vp = [_ for _ in chain.from_iterable(v[k] + t*u[k] for k in K[:-1])]

    fig = plt.figure(figsize=(10,5))

    ax1 = fig.add_subplot(3, 2, 1)
    ax1.plot(tp, yp, 'r--', h*K, y, 'bo')
    ax1.set_title('position')

    ax2 = fig.add_subplot(3, 2, 3)
    ax2.plot(tp, vp, 'r--', h*K, v, 'bo')
    ax2.set_title('velocity')

    ax3 = fig.add_subplot(3, 2, 5)
    ax3.plot(np.append(tp, K[-1]*h), np.append(up, u[-1]), 'r--', h*K, u, 'bo')
    ax3.set_title('control force  u[0] = {0:<6.3f}'.format(u[0]))

    ax4 = fig.add_subplot(1, 2, 2)
    ax4.plot(yp, vp, 'r--', y, v, 'bo')
    ax4.set_xlim([-1.1, 1.1])
    ax4.set_aspect('equal', 'box')
    ax4.set_title('phase plane')
    ax4.set_xlabel('position')
    ax4.set_ylabel('velocity')

    for ax in [ax1, ax2, ax3, ax4]:
        ax.set_ylim(-1.1, 1.1)
        ax.grid(True)
    fig.tight_layout()

model = mpc_double_integrator(5, 0.5)
model.ic[1] = 1.0
model.ic[2] = 0.2

SolverFactory('cbc').solve(model)
plot_results(model)
```

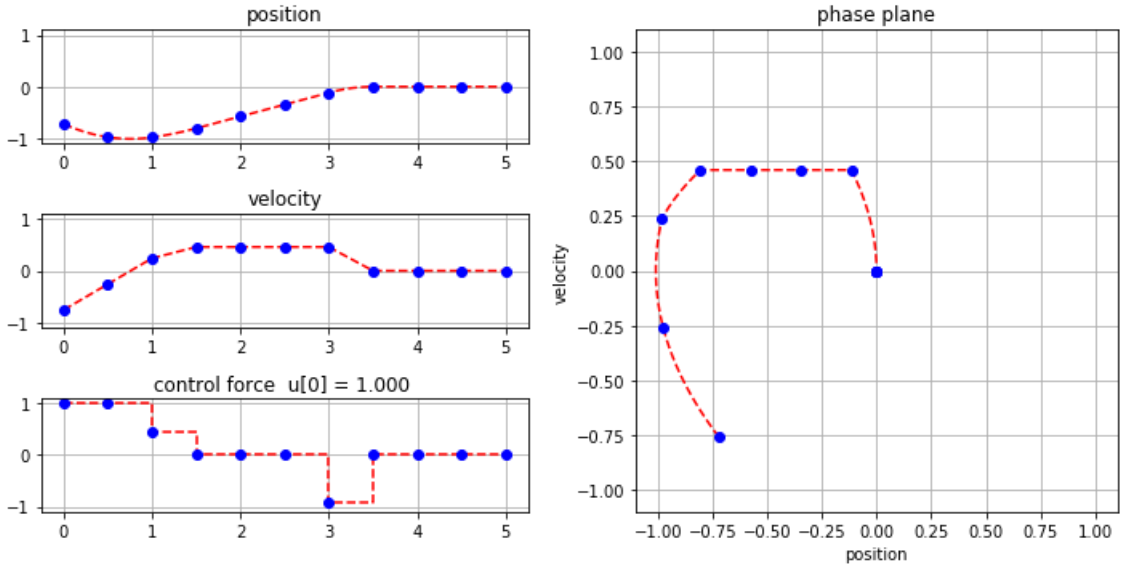


Interactive Use

Google Colab

In [0]:

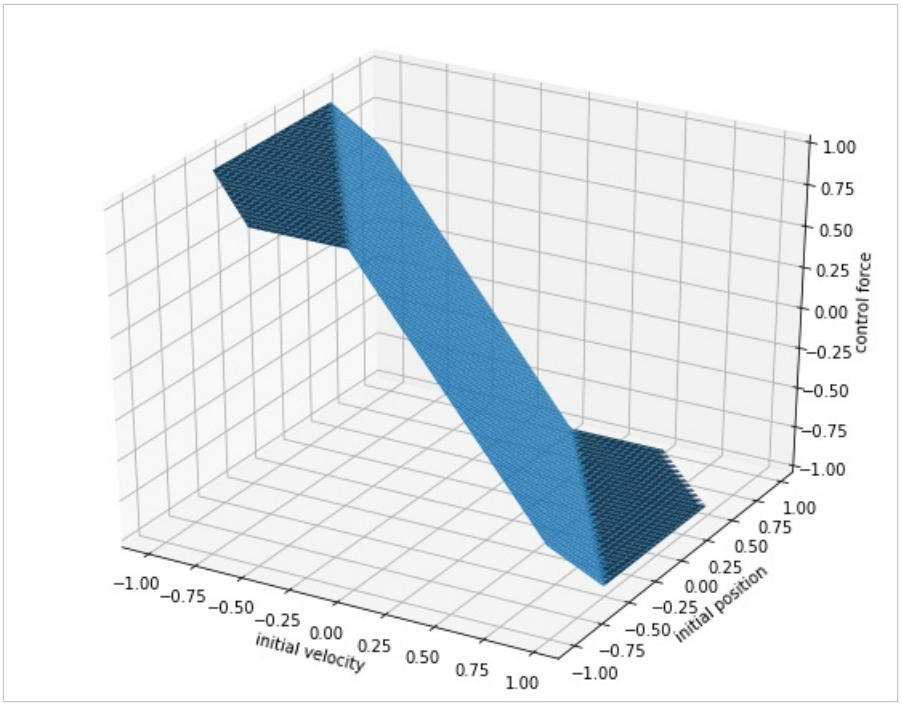
```
#@title Interactive { run: "auto" }

N = 10 #@param {type: "slider", min:1, max:20, step:1}
h = 0.5 #@param {type:"slider", min:0, max:1, step:0.01}
gamma = 0.54 #@param {type:"slider", min:0, max:1, step:0.01}
x_initial = -0.72 #@param {type:"slider", min:-1, max:1, step:0.01}
v_initial = -0.76 #@param {type:"slider", min:-1, max:1, step:0.01}

model = mpc_double_integrator(N, h)
model.gamma = gamma
model.ic[1] = x_initial
model.ic[2] = v_initial

SolverFactory('cbc').solve(model)
plot_results(model)
```



Model Predictive Control as a Feedback Controller

In [0]:

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import random

model = mpc_double_integrator(5)
model.h = 1
model.gamma = 0.4

def fun(y, v):
    u = 0*y
    for i in range(0, len(y)):
        model.ic[1] = y[i]
        model.ic[2] = v[i]
        results = solver.solve(model)
        if str(results.solver.termination_condition) == 'optimal':
            u[i] = model.u[0]()
        else:
            u[i] = None
    return u

fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111, projection='3d')

y = v = np.arange(-1, 1.0, 0.02)
Y, V = np.meshgrid(y, v)
u = np.array(fun(np.ravel(Y), np.ravel(V)))
U = u.reshape(V.shape)

ax.plot_surface(V, Y, U)

ax.set_xlabel('initial velocity')
ax.set_ylabel('initial position')
ax.set_zlabel('control force')

plt.show()
```



In [0]:

```
```

Chapter 3. Assignment Problems

3.1 Transportation Networks

Background

The prototypical transportation problem deals with the distribution of a commodity from a set of sources to a set of destinations. The object is to minimize total transportation costs while satisfying constraints on the supplies available at each of the sources, and satisfying demand requirements at each of the destinations.

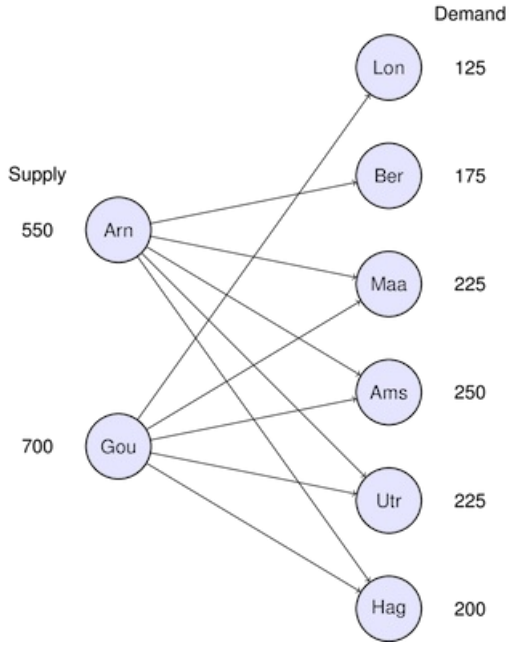
Here we illustrate the transportation problem using an example from Chapter 5 of Johannes Bisschop, "AIMMS Optimization Modeling", Paragon Decision Sciences, 1999. In this example there are two factories and six customer sites located in 8 European cities as shown in the following map. The customer sites are labeled in red, the factories are labeled in blue.

Transportation costs between sources and destinations are given in units of €/ton of goods shipped, and list in the following table along with source capacity and demand requirements.

Table of Transportation Costs, Customer Demand, and Available Supplies

Customer\Source	Arnhem [€/ton]	Gouda [€/ton]	Demand [tons]
London	n/a	2.5	125
Berlin	2.5	n/a	175
Maastricht	1.6	2.0	225
Amsterdam	1.4	1.0	250
Utrecht	0.8	1.0	225
The Hague	1.4	0.8	200
Supply [tons]	550 tons	700 tons	

The situation can be modeled by links connecting a set nodes representing sources to a set of nodes representing customers.



For each link we can have a parameter $T[c, s]$ denoting the cost of shipping a ton of goods over the link. What we need to determine is the amount of goods to be shipped over each link, which we will represent as a non-negative decision variable $x[c, s]$.

The problem objective is to minimize the total shipping cost to all customers from all sources.

$$\text{minimize:} \quad \text{Cost} = \sum_{c \in Customers} \sum_{s \in Sources} T[c, s] x[c, s]$$

Shipments from all sources can not exceed the manufacturing capacity of the source.

$$\sum_{c \in Customers} x[c, s] \leq \text{Supply}[s] \qquad \forall s \in Sources$$

Shipments to each customer must satisfy their demand.

$$\sum_{s \in Sources} x[c, s] = \text{Demand}[c] \qquad \forall c \in Customers$$

Pyomo Model

In [ ]:

```
%%capture
!pip install -q pyomo
!apt-get install -y -qq coinor-cbc
```

Data File

In [ ]:

```
Demand = {
    'Lon':   125,         # London
    'Ber':   175,         # Berlin
    'Maa':   225,         # Maastricht
    'Ams':   250,         # Amsterdam
    'Utr':   225,         # Utrecht
    'Hag':   200          # The Hague
}

Supply = {
    'Arn':   600,         # Arnhem
    'Gou':   650          # Gouda
}

T = {
     ('Lon','Arn'): 1000,
     ('Lon','Gou'): 2.5,
     ('Ber','Arn'): 2.5,
     ('Ber','Gou'): 1000,
     ('Maa','Arn'): 1.6,
     ('Maa','Gou'): 2.0,
     ('Ams','Arn'): 1.4,
     ('Ams','Gou'): 1.0,
     ('Utr','Arn'): 0.8,
     ('Utr','Gou'): 1.0,
     ('Hag','Arn'): 1.4,
     ('Hag','Gou'): 0.8
}
```

Model File

In [3]:

```
from pyomo.environ import *

# Step 0: Create an instance of the model
model = ConcreteModel()
model.dual = Suffix(direction=Suffix.IMPORT)

# Step 1: Define index sets
CUS = list(Demand.keys())
SRC = list(Supply.keys())

# Step 2: Define the decision
model.x = Var(CUS, SRC, domain = NonNegativeReals)

# Step 3: Define Objective
model.Cost = Objective(
    expr = sum([T[c,s]*model.x[c,s] for c in CUS for s in SRC]),
    sense = minimize)

# Step 4: Constraints
model.src = ConstraintList()
for s in SRC:
    model.src.add(sum([model.x[c,s] for c in CUS]) <= Supply[s])

model.dmd = ConstraintList()
for c in CUS:
    model.dmd.add(sum([model.x[c,s] for s in SRC]) == Demand[c])

results = SolverFactory('cbc').solve(model)
results.write()
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: 1705.0
  Upper bound: 1705.0
  Number of objectives: 1
  Number of constraints: 9
  Number of variables: 13
  Number of nonzeros: 25
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  User time: -1.0
  Termination condition: optimal
  Error rc: 0
  Time: 0.016471385955810547
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

Solution

In [4]:

```
for c in CUS:
    for s in SRC:
        print(c, s, model.x[c,s]())
```

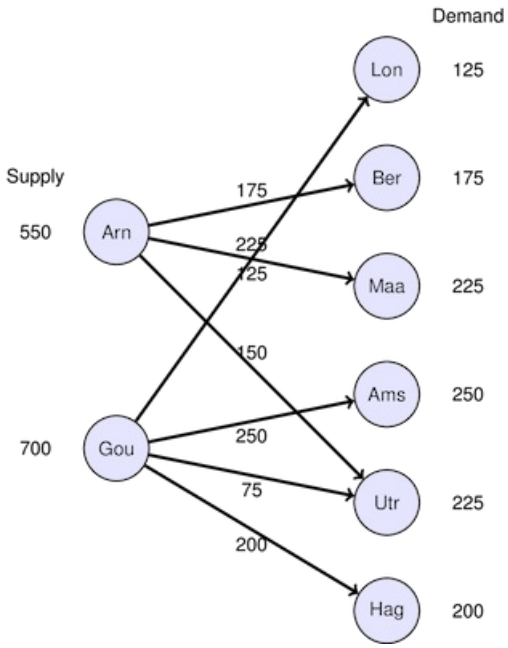
```
Lon Arn 0.0
Lon Gou 125.0
Ber Arn 175.0
Ber Gou 0.0
Maa Arn 225.0
Maa Gou 0.0
Ams Arn 0.0
Ams Gou 250.0
Utr Arn 200.0
Utr Gou 25.0
Hag Arn 0.0
Hag Gou 200.0
```

In [5]:

```
if 'ok' == str(results.Solver.status):
    print("Total Shipping Costs = ",model.Cost())
    print("\nShipping Table:")
    for s in SRC:
        for c in CUS:
            if model.x[c,s]() > 0:
                print("Ship from ", s," to ", c, ":", model.x[c,s]())
else:
    print("No Valid Solution Found")
```

```
Total Shipping Costs =  1705.0

Shipping Table:
Ship from  Arn  to  Ber : 175.0
Ship from  Arn  to  Maa : 225.0
Ship from  Arn  to  Utr : 200.0
Ship from  Gou  to  Lon : 125.0
Ship from  Gou  to  Ams : 250.0
Ship from  Gou  to  Utr : 25.0
Ship from  Gou  to  Hag : 200.0
```

The solution has the interesting property that, with the exception of Utrecht, customers are served by just one source.



Sensitivity Analysis

Analysis by Source

In [6]:

```
if 'ok' == str(results.Solver.status):
    print("\nSources:")
    print("Source      Capacity   Shipped    Margin")
    for m in model.src.keys():
        s = SRC[m-1]
        print("{0:10s}{1:10.1f}{2:10.1f}{3:10.4f}".format(s,Supply[s],model.src[m](),mo
del.dual[model.src[m]]))
else:
    print("No Valid Solution Found")
```

```
Sources:
Source      Capacity   Shipped    Margin
Arn            600.0     600.0   -0.2000
Gou            650.0     600.0    0.0000
```

The 'marginal' values are telling us how much the total costs will be increased for each one ton increase in the available supply from each source. The optimization calculation says that only 650 tons of the 700 available from Gouda should used for a minimum cost solution, which rules out any further cost reductions by increasing the available supply. In fact, we could decrease the supply Gouda without any harm. The marginal value of Gouda is 0.

The source at Arnhem is a different matter. First, all 550 available tons are being used. Second, from the marginal value we see that the total transportations costs would be reduced by 0.2 Euros for each additional ton of supply.

The management conclusion we can draw is that there is excess supply available at Gouda which should, if feasible, me moved to Arnhem.

Now that's a valuable piece of information!

Analysis by Customer

In [7]:

```
if 'ok' == str(results.Solver.status):
    print("\nCustomers:")
    print("Customer      Demand   Shipped    Margin")
    for n in model.dmd.keys():
        c = CUS[n-1]
        print("{0:10s}{1:10.1f}{2:10.1f}{3:10.4f}".format(c,Demand[c],model.dmd[n](),mo
del.dual[model.dmd[n]]))
else:
    print("No Valid Solution Found")
```

```
Customers:
Customer      Demand   Shipped    Margin
Lon            125.0     125.0    2.5000
Ber            175.0     175.0    2.7000
Maa            225.0     225.0    1.8000
Ams            250.0     250.0    1.0000
Utr            225.0     225.0    1.0000
Hag            200.0     200.0    0.8000
```

Looking at the demand constraints, we see that all of the required demands have been met by the optimal solution.

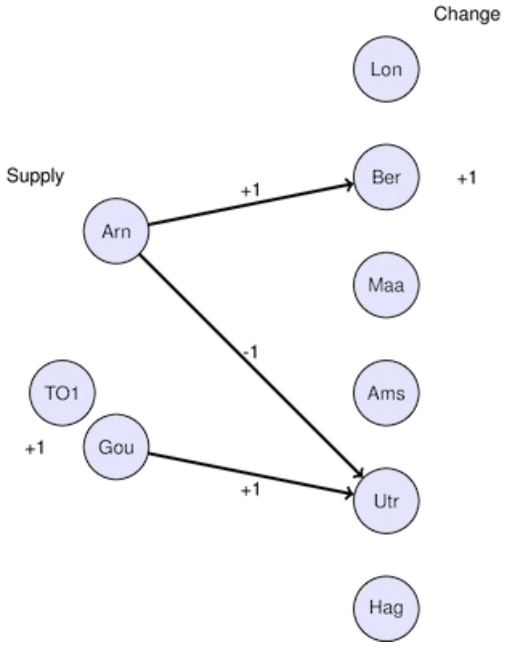
The marginal values of these constraints indicate how much the total transportation costs will increase if there is an additional ton of demand at any of the locations. In particular, note that increasing the demand at Berlin will increase costs by 2.7 Euros per ton. This is actually **greater** than the list price for shipping to Berlin which is 2.5 Euros per ton. Why is this?

To see what's going on, let's resolve the problem with a one ton increase in the demand at Berlin.

We see the total cost has increased from 1715.0 to 1717.7 Euros, an increase of 2.7 Euros just as predicted by the marginal value assocated with the demand constraint for Berlin.

Now let's look at the solution.

Here we see that increasing the demand in Berlin resulted in a number of other changes. This figure shows the changes shipments.



- Shipments to Berlin increased from 175 to 176 tons, increasing costs for that link from 437.5 to 440.0, or a net increase of 2.5 Euros.
- Since Arnhem is operating at full capacity, increasing the shipments from Arnhem to Berlin resulted in decreasing the shipments from Arhhem to Utrecht from 150 to 149 reducing those shipping costs from 120.0 to 119.2, a net decrease of 0.8 Euros.
- To meet demand at Utrecht, shipments from Gouda to Utrecht had to increase from 75 to 76, increasing shipping costs by a net amount of 1.0 Euros.
- The net effect on shipping costs is 2.5 - 0.8 + 1.0 = 2.7 Euros.

The important conclusion to draw is that when operating under optimal conditions, a change in demand or supply can have a ripple effect on the optimal solution that can only be measured through a proper sensitivity analysis.

Exercises

1. Move 50 tons of supply capacity from Gouda to Arnhem, and repeat the sensitivity analysis. How has the situation improved? In practice, would you recommend this change, or would you propose something different?
2. What other business improvements would you recommend?

In [ ]:

```
```

Chapter 4. Scheduling with Disjunctive Constraints

4.1 Machine Bottleneck

In [ ]:

```
%%capture
!pip install -q pyomo
!apt-get install -y -qq coinor-cbc
```

Example

The problem is to schedule a sequence of jobs for a single machine. The data consists of a Python dictionary of jobs. Each job is labeled by a key, and an associated data dictionary provides the time at which the job is released to the for machine processing, the expected duration of the job, and the due date. The problem is to sequence the jobs on the machine to meet the due dates, or show that no such sequence is possible.

In [2]:

```
JOBS = {
    'A': {'release': 2, 'duration': 5, 'due': 10},
    'B': {'release': 5, 'duration': 6, 'due': 21},
    'C': {'release': 4, 'duration': 8, 'due': 15},
    'D': {'release': 0, 'duration': 4, 'due': 10},
    'E': {'release': 0, 'duration': 2, 'due':  5},
    'F': {'release': 8, 'duration': 3, 'due': 15},
    'G': {'release': 9, 'duration': 2, 'due': 22},
}
JOBS
```

Out[2]:

```
{'A': {'due': 10, 'duration': 5, 'release': 2},
 'B': {'due': 21, 'duration': 6, 'release': 5},
 'C': {'due': 15, 'duration': 8, 'release': 4},
 'D': {'due': 10, 'duration': 4, 'release': 0},
 'E': {'due': 5, 'duration': 2, 'release': 0},
 'F': {'due': 15, 'duration': 3, 'release': 8},
 'G': {'due': 22, 'duration': 2, 'release': 9}}
```

The Machine Scheduling Problem

A schedule consists of a dictionary listing the start and finish times for each job. Once the order of jobs has been determined, the start time can be no earlier than when the job is released for processing, and no earlier than the finish of the previous job.

The following cell presents a function which, given the JOBS data and an order list of jobs indices, computes the start and finish times for all jobs on a single machine. We use this to determine the schedule if the jobs are executed in alphabetical order.

In [3]:

```
def schedule(JOBS, order=sorted(JOBS.keys())):
    """Schedule a dictionary of JOBS on a single machine in a specified order."""
    start = 0
    finish = 0
    SCHEDULE = {}
    for job in order:
        start = max(JOBS[job]['release'], finish)
        finish = start + JOBS[job]['duration']
        SCHEDULE[job] = {'start': start, 'finish': finish, }
    return SCHEDULE

SCHEDULE = schedule(JOBS)
SCHEDULE
```

Out[3]:

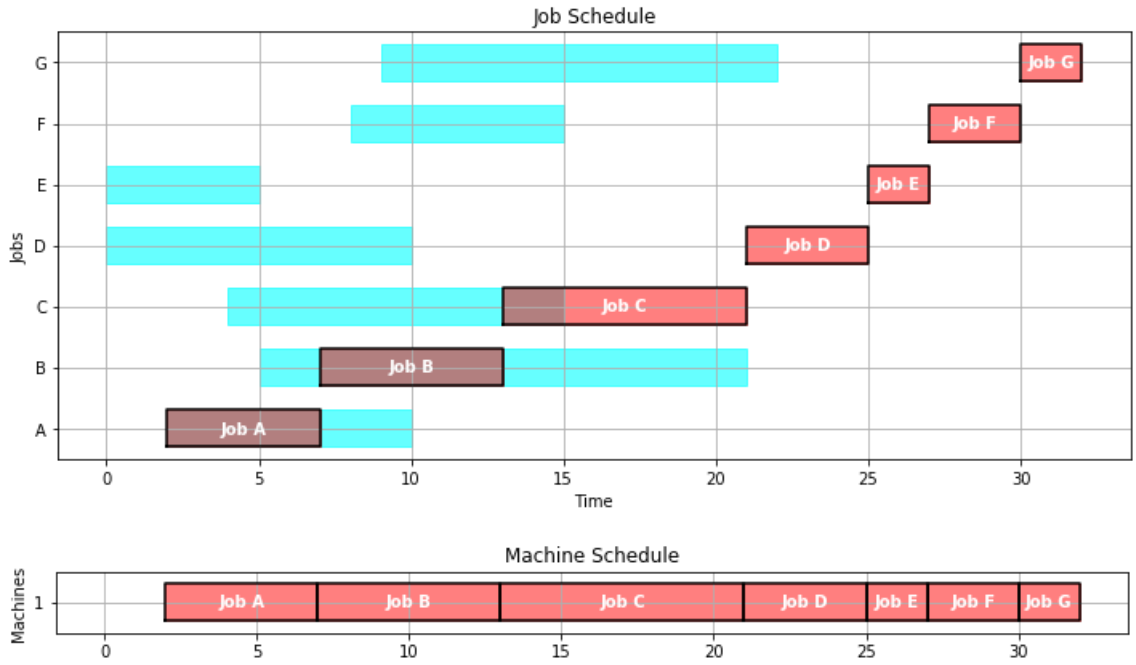
```
{'A': {'finish': 7, 'start': 2},
 'B': {'finish': 13, 'start': 7},
 'C': {'finish': 21, 'start': 13},
 'D': {'finish': 25, 'start': 21},
 'E': {'finish': 27, 'start': 25},
 'F': {'finish': 30, 'start': 27},
 'G': {'finish': 32, 'start': 30}}
```

Gantt Chart

A traditional means of visualizing scheduling data in the form of a Gantt chart. The next cell presents a function `gantt` that plots a Gantt chart given JOBS and SCHEDULE information. Two charts are presented showing job schedule and machine schedule. If no machine information is contained in SCHEDULE, then it assumed to be a single machine operation.

In [4]:

```
%matplotlib inline
import matplotlib.pyplot as plt

def gantt(JOBS, SCHEDULE):
    bw = 0.3
    plt.figure(figsize=(12, 0.7*(len(JOBS.keys()))))
    idx = 0
    for j in sorted(JOBS.keys()):
        x = JOBS[j]['release']
        y = JOBS[j]['due']
        plt.fill_between([x,y],[idx-bw,idx-bw],[idx+bw,idx+bw], color='cyan', alpha=0.6
)
        x = SCHEDULE[j]['start']
        y = SCHEDULE[j]['finish']
        plt.fill_between([x,y],[idx-bw,idx-bw],[idx+bw,idx+bw], color='red', alpha=0.5)
        plt.plot([x,y,y,x,x], [idx-bw,idx-bw,idx+bw,idx+bw,idx-bw],color='k')
        plt.text((SCHEDULE[j]['start'] + SCHEDULE[j]['finish'])/2.0,idx,
            'Job ' + j, color='white', weight='bold',
            horizontalalignment='center', verticalalignment='center')
        idx += 1

    plt.ylim(-0.5, idx-0.5)
    plt.title('Job Schedule')
    plt.xlabel('Time')
    plt.ylabel('Jobs')
    plt.yticks(range(len(JOBS)), JOBS.keys())
    plt.grid()
    xlim = plt.xlim()

    for j in JOBS.keys():
        if 'machine' not in SCHEDULE[j].keys():
            SCHEDULE[j]['machine'] = 1
    MACHINES = sorted(set([SCHEDULE[j]['machine'] for j in JOBS.keys()]))

    plt.figure(figsize=(12, 0.7*len(MACHINES)))
    for j in sorted(JOBS.keys()):
        idx = MACHINES.index(SCHEDULE[j]['machine'])
        x = SCHEDULE[j]['start']
        y = SCHEDULE[j]['finish']
        plt.fill_between([x,y],[idx-bw,idx-bw],[idx+bw,idx+bw], color='red', alpha=0.5)
        plt.plot([x,y,y,x,x], [idx-bw,idx-bw,idx+bw,idx+bw,idx-bw],color='k')
        plt.text((SCHEDULE[j]['start'] + SCHEDULE[j]['finish'])/2.0,idx,
            'Job ' + j, color='white', weight='bold',
            horizontalalignment='center', verticalalignment='center')
    plt.xlim(xlim)
    plt.ylim(-0.5, len(MACHINES)-0.5)
    plt.title('Machine Schedule')
    plt.yticks(range(len(MACHINES)), MACHINES)
    plt.ylabel('Machines')
    plt.grid()

gantt(JOBS, SCHEDULE)
```



Key Performance Indicators

As presented above, a given schedule may not meet all of the due time requirements. In fact, a schedule meeting all of the requirements might not even be possible. So given a schedule, it is useful to have a function that computes key performance indicators.

In [5]:

```
def kpi(JOBS, SCHEDULE):
    KPI = {}
    KPI['Makespan'] = max(SCHEDULE[job]['finish'] for job in JOBS)
    KPI['Max Pastdue'] = max(max(0, SCHEDULE[job]['finish'] - JOBS[job]['due']) for job
in JOBS)
    KPI['Sum of Pastdue'] = sum(max(0, SCHEDULE[job]['finish'] - JOBS[job]['due']) for
job in JOBS)
    KPI['Number Pastdue'] = sum(SCHEDULE[job]['finish'] > JOBS[job]['due'] for job in J
OBS)
    KPI['Number on Time'] = sum(SCHEDULE[job]['finish'] <= JOBS[job]['due'] for job in
JOBS)
    KPI['Fraction on Time'] = KPI['Number on Time']/len(JOBS)
    return KPI

kpi(JOBS, SCHEDULE)
```

Out[5]:

```
{'Fraction on Time': 0.2857142857142857,
 'Makespan': 32,
 'Max Pastdue': 22,
 'Number Pastdue': 5,
 'Number on Time': 2,
 'Sum of Pastdue': 68}
```

Exercise

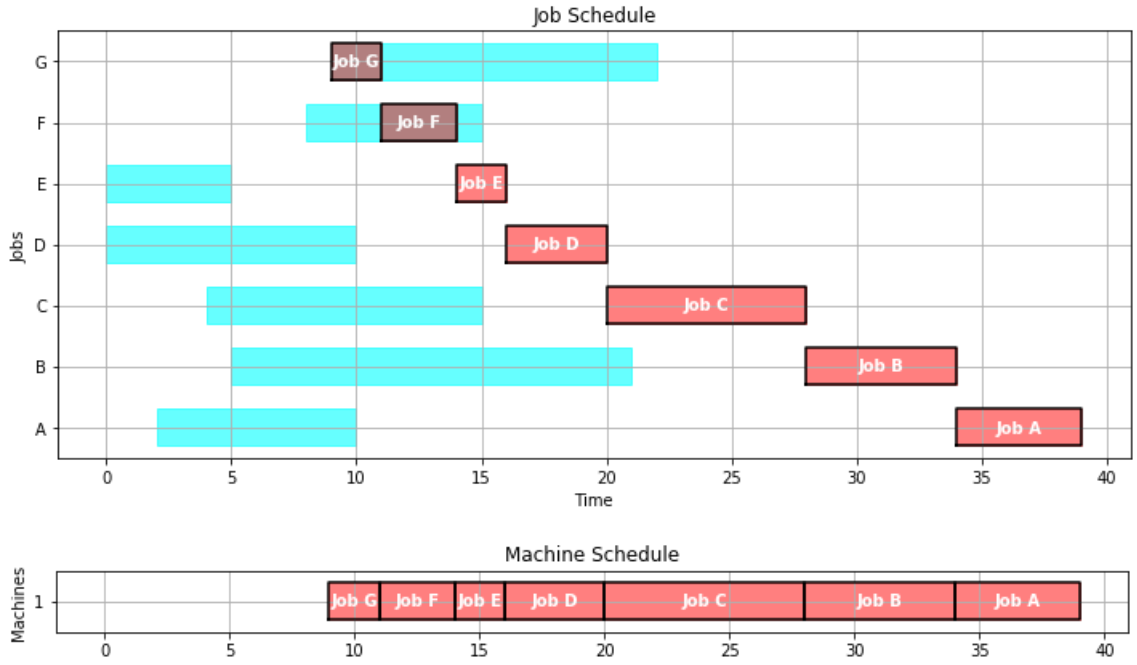
Show the Gantt chart and key performance metrics if the jobs are executed in reverse alphabetical order.

In [6]:

```
order = sorted(JOBS, reverse=True)
gantt(JOBS, schedule(JOBS,order))
kpi(JOBS, schedule(JOBS,order))
```

Out[6]:

```
{'Fraction on Time': 0.2857142857142857,
 'Makespan': 39,
 'Max Pastdue': 29,
 'Number Pastdue': 5,
 'Number on Time': 2,
 'Sum of Pastdue': 76}
```



Empirical Scheduling

There are a number of commonly encountered empirical rules for scheduling jobs on a single machine. These include:

- First-In First-Out (FIFO)
- Last-In, First-Out (LIFO)
- Shortest Processing Time First (SPT)
- Earliest Due Data (EDD)

First-In First-Out

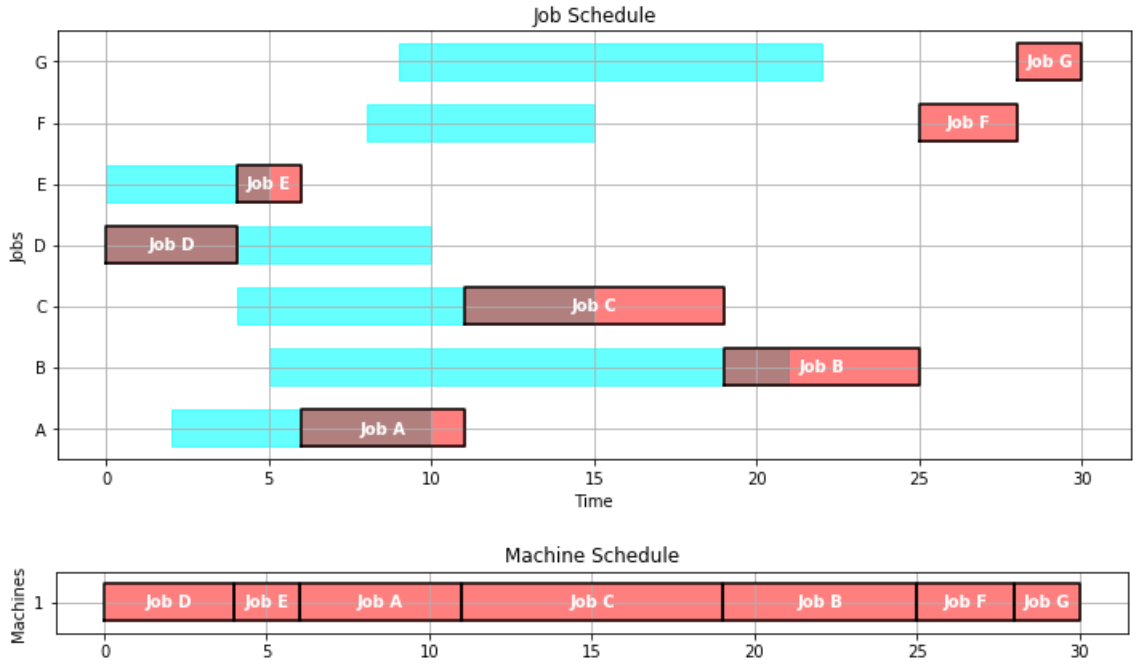
As an example, we'll first look at 'First-In-First-Out' scheduling which executes job in the order they are released. The following function sorts jobs by release time, then schedules the jobs to execute in that order. A job can only be started no earlier than when it is released.

In [7]:

```
def fifo(JOBS):
    order_by_release = sorted(JOBS, key=lambda job: JOBS[job]['release'])
    return schedule(JOBS, order_by_release)

SCHEDULE = fifo(JOBS)
gantt(JOBS, SCHEDULE)
kpi(JOBS, SCHEDULE)
```

Out[7]:

```
{'Fraction on Time': 0.14285714285714285,
 'Makespan': 30,
 'Max Pastdue': 13,
 'Number Pastdue': 6,
 'Number on Time': 1,
 'Sum of Pastdue': 31}
```



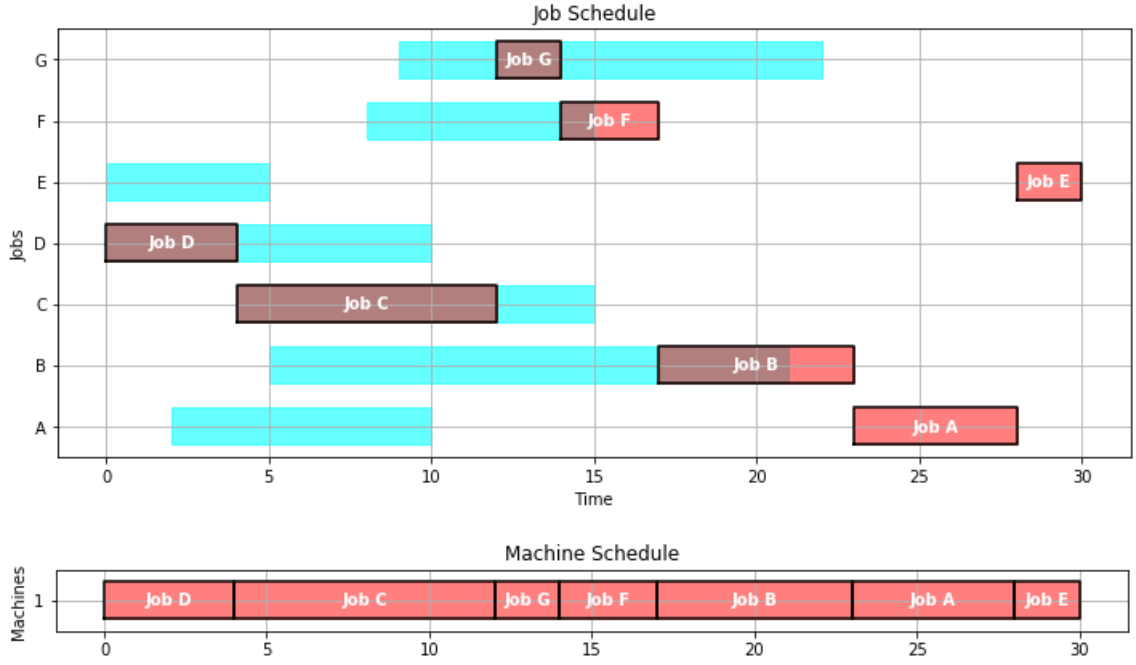
Last-In, First-Out

In [8]:

```
def lifo(JOBS):
    unfinished_jobs = set(JOBS.keys())
    start = 0
    while len(unfinished_jobs) > 0:
        start = max(start, min(JOBS[job]['release'] for job in unfinished_jobs))
        lifo = {job:JOBS[job]['release'] for job in unfinished_jobs if JOBS[job]
['release'] <= start}
        job = max(lifo, key=lifo.get)
        finish = start + JOBS[job]['duration']
        unfinished_jobs.remove(job)
        SCHEDULE[job] = {'machine': 1, 'start': start, 'finish': finish}
        start = finish
    return SCHEDULE

gantt(JOBS, lifo(JOBS))
kpi(JOBS, lifo(JOBS))
```

Out[8]:

```
{'Fraction on Time': 0.42857142857142855,
 'Makespan': 30,
 'Max Pastdue': 25,
 'Number Pastdue': 4,
 'Number on Time': 3,
 'Sum of Pastdue': 47}
```



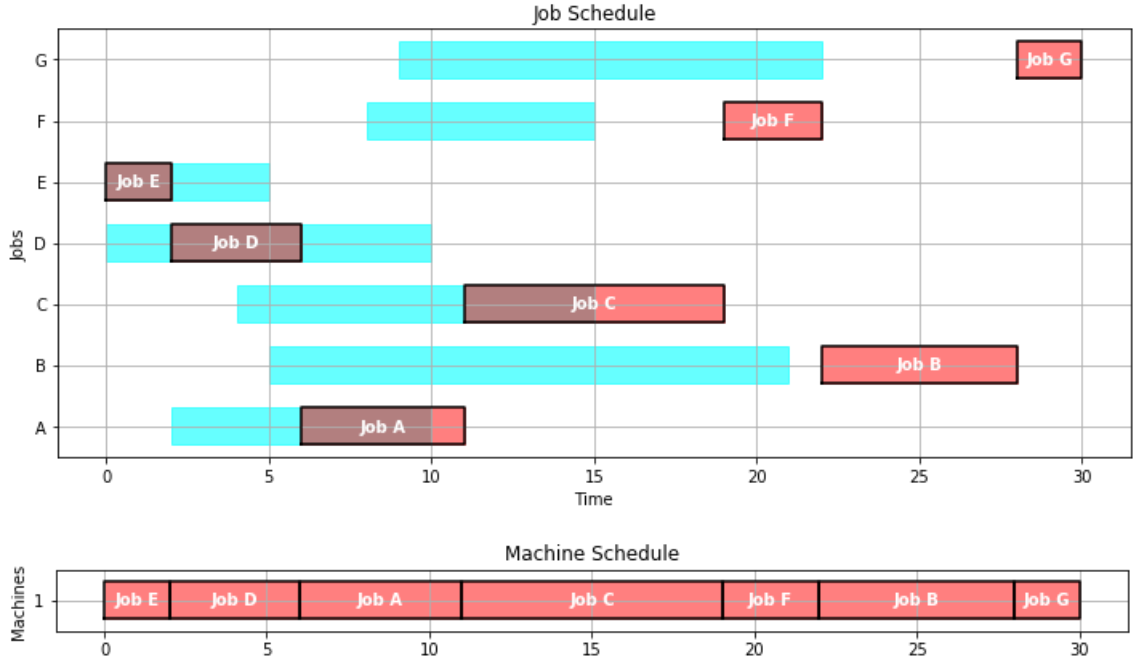
Earliest Due Date

In [9]:

```
def edd(JOBS):
    unfinished_jobs = set(JOBS.keys())
    start = 0
    while len(unfinished_jobs) > 0:
        start = max(start, min(JOBS[job]['release'] for job in unfinished_jobs))
        edd = {job:JOBS[job]['due'] for job in unfinished_jobs if JOBS[job]['release']
<= start}
        job = min(edd, key=edd.get)
        finish = start + JOBS[job]['duration']
        unfinished_jobs.remove(job)
        SCHEDULE[job] = {'machine': 1, 'start': start, 'finish': finish}
        start = finish
    return SCHEDULE

gantt(JOBS, edd(JOBS))
kpi(JOBS, edd(JOBS))
```

Out[9]:

```
{'Fraction on Time': 0.2857142857142857,
 'Makespan': 30,
 'Max Pastdue': 8,
 'Number Pastdue': 5,
 'Number on Time': 2,
 'Sum of Pastdue': 27}
```



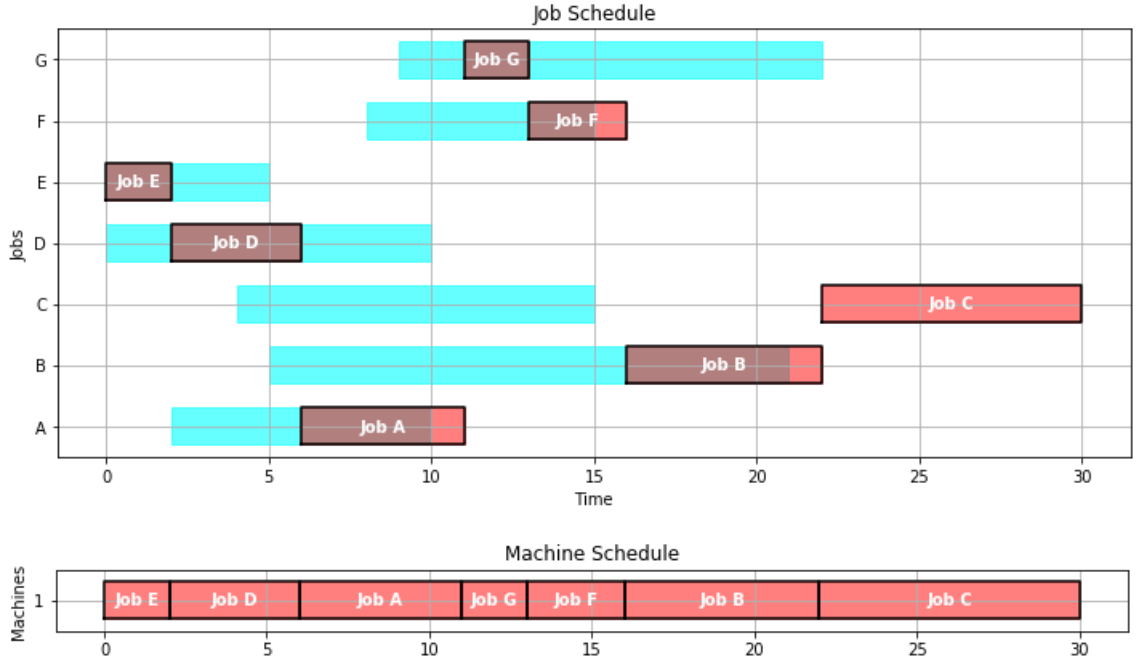
Shortest Processing Time

In [10]:

```
def spt(JOBS):
    unfinished_jobs = set(JOBS.keys())
    start = 0
    while len(unfinished_jobs) > 0:
        start = max(start, min(JOBS[job]['release'] for job in unfinished_jobs))
        spt = {job:JOBS[job]['duration'] for job in unfinished_jobs if JOBS[job]
['release'] <= start}
        job = min(spt, key=spt.get)
        finish = start + JOBS[job]['duration']
        unfinished_jobs.remove(job)
        SCHEDULE[job] = {'machine': 1, 'start': start, 'finish': finish}
        start = finish
    return SCHEDULE

gantt(JOBS, spt(JOBS))
kpi(JOBS, spt(JOBS))
```

Out[10]:

```
{'Fraction on Time': 0.42857142857142855,
 'Makespan': 30,
 'Max Pastdue': 15,
 'Number Pastdue': 4,
 'Number on Time': 3,
 'Sum of Pastdue': 18}
```



Modeling

Data

The data for this problem consists of a list of jobs. Each job is tagged with a unique ID along with numerical data giving the time at which the job will be released for machine processing, the expected duration, and the time at which it is due.

Symbol	Description
ID_j	Unique ID for task j
due_j	Due time for task j
duration_j	Duration of task j
release_j	Time task j becomes available for processing

Decision Variables

For a single machine, the essential decision variable is the start time at which the job begins processing.

Symbol	Description
start_j	Start of task j
makespan	Time to complete *all* jobs.
pastdue_j	Time by which task j is past due
early_j	Time by which task j is finished early

A job cannot start until it is released for processing

$$\text{start}_j \geq \text{release}_j$$

Once released for processing, we assume the processing continues until the job is finished. The finish time is compared to the due time, and the result stored in either the early or pastdue decision variables. These decision variables are needed to handle cases where it might not be possible to complete all jobs by the time they are due.

$$\begin{aligned} \text{start}_j + \text{duration}_j + \text{early}_j &= \text{due}_j + \text{pastdue}_j \\ \text{early}_j &\geq 0 \\ \text{pastdue}_j &\geq 0 \end{aligned}$$

Finally, we include a single decision variable measuring the overall makespan for all jobs.

$$\text{start}_j + \text{duration}_j \leq \text{makespan}$$

The final set of constraints requires that, for any given pair of jobs j and k, that either j starts before k finishes, or k finishes before j starts. The boolean variable $y_{jk} = 1$ indicates j finishes before k starts, and is 0 for the opposing case. Note that we only need to consider cases $j > k$

$$\begin{aligned} \text{start}_i + \text{duration}_i &\leq \text{start}_j + My_{i,j} \\ \text{start}_j + \text{duration}_j &\leq \text{start}_i + M(1 - y_{i,j}) \end{aligned}$$

where M is a sufficiently large enough to assure the relaxed constraint is satisfied for all plausible values of the decision variables.

Pyomo Model

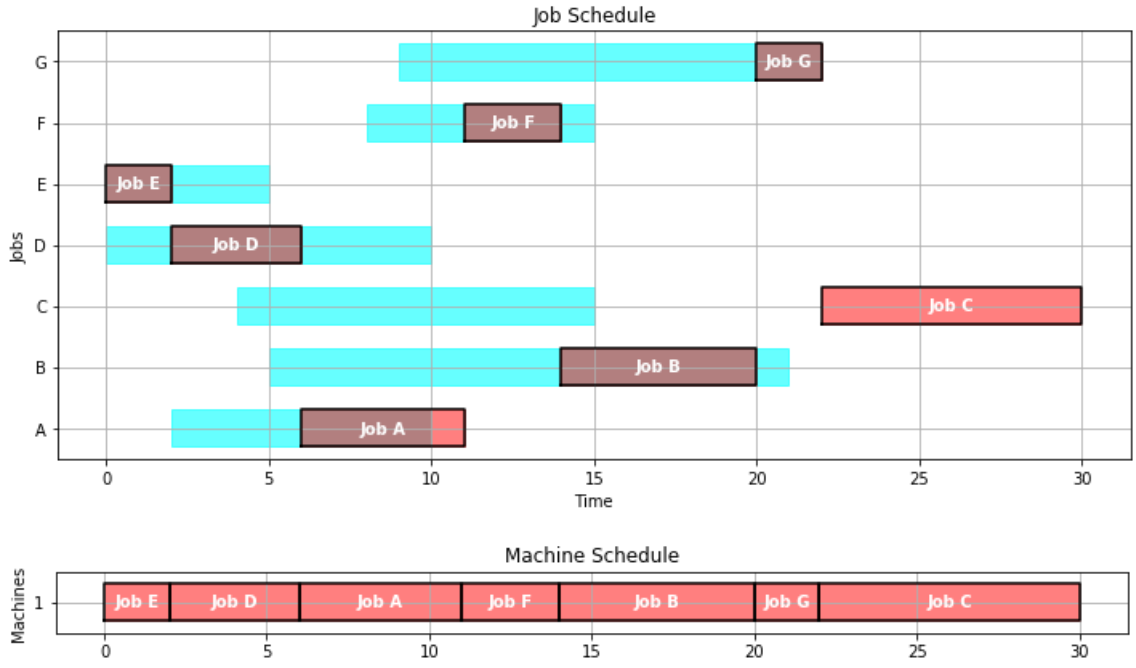
In [11]:

```
from pyomo.environ import *
from pyomo.gdp import *

def opt_schedule(JOBS):

    # create model
    m = ConcreteModel()

    # index set to simplify notation
    m.J = Set(initialize=JOBS.keys())
    m.PAIRS = Set(initialize = m.J * m.J, dimen=2, filter=lambda m, j, k : j < k)

    # upper bounds on how long it would take to process all jobs
    tmax = max([JOBS[j]['release'] for j in m.J]) + sum([JOBS[j]['duration'] for j in m.
J])

    # decision variables
    m.start      = Var(m.J, domain=NonNegativeReals, bounds=(0, tmax))
    m.pastdue    = Var(m.J, domain=NonNegativeReals, bounds=(0, tmax))
    m.early      = Var(m.J, domain=NonNegativeReals, bounds=(0, tmax))

    # additional decision variables for use in the objecive
    m.makespan   = Var(domain=NonNegativeReals, bounds=(0, tmax))
    m.maxpastdue = Var(domain=NonNegativeReals, bounds=(0, tmax))
    m.ispastdue  = Var(m.J, domain=Binary)

    # objective function
    m.OBJ = Objective(expr = sum([m.pastdue[j] for j in m.J]), sense = minimize)

    # constraints
    m.c1 = Constraint(m.J, rule=lambda m, j: m.start[j] >= JOBS[j]['release'])
    m.c2 = Constraint(m.J, rule=lambda m, j:
            m.start[j] + JOBS[j]['duration'] + m.early[j] == JOBS[j]['due'] + m.pastdue[
j])
    m.c3 = Disjunction(m.PAIRS, rule=lambda m, j, k:
        [m.start[j] + JOBS[j]['duration'] <= m.start[k],
         m.start[k] + JOBS[k]['duration'] <= m.start[j]])

    m.c4 = Constraint(m.J, rule=lambda m, j: m.pastdue[j] <= m.maxpastdue)
    m.c5 = Constraint(m.J, rule=lambda m, j: m.start[j] + JOBS[j]['duration'] <= m.makes
pan)
    m.c6 = Constraint(m.J, rule=lambda m, j: m.pastdue[j] <= tmax*m.ispastdue[j])

    TransformationFactory('gdp.chull').apply_to(m)
    SolverFactory('cbc').solve(m).write()

    SCHEDULE = {}
    for j in m.J:
        SCHEDULE[j] = {'machine': 1, 'start': m.start[j](), 'finish': m.start[j]() + JOB
S[j]['duration']}

    return SCHEDULE

SCHEDULE = opt_schedule(JOBS)
gantt(JOBS, SCHEDULE)
kpi(JOBS, SCHEDULE)
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: 16.0
  Upper bound: 16.0
  Number of objectives: 1
  Number of constraints: 225
  Number of variables: 157
  Number of nonzeros: 533
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  User time: -1.0
  Termination condition: optimal
  Error rc: 0
  Time: 0.56770920753479
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

Out[11]:

```
{'Fraction on Time': 0.7142857142857143,
 'Makespan': 30.0,
 'Max Pastdue': 15.0,
 'Number Pastdue': 2,
 'Number on Time': 5,
 'Sum of Pastdue': 16.0}
```



Multiple Machines

The case of multiple machines requires a modest extension of model described above. Given a set M of machines, we introduce an additional decision binary variable $z_{j,m}$ indicating if job j has been assigned to machine m. The additional constraints

$$\sum_{m \in M} z_{j,m} = 1 \qquad \forall j$$

require each job to be assigned to exactly one machine for processing.

If both jobs j and k have been assigned to machine m, then the disjunctive ordering constraints must apply. This logic is equivalent to the following constraints for $j < k$.

$$\begin{aligned} \text{start}_j + \text{duration}_j &\leq \text{start}_k + My_{j,k} + M(1 - z_{j,m}) + M(1 - z_{k,m}) \\ \text{start}_k + \text{duration}_k &\leq \text{start}_j + M(1 - y_{j,k}) + M(1 - z_{j,m}) + M(1 - z_{k,m})) \end{aligned}$$

In [12]:

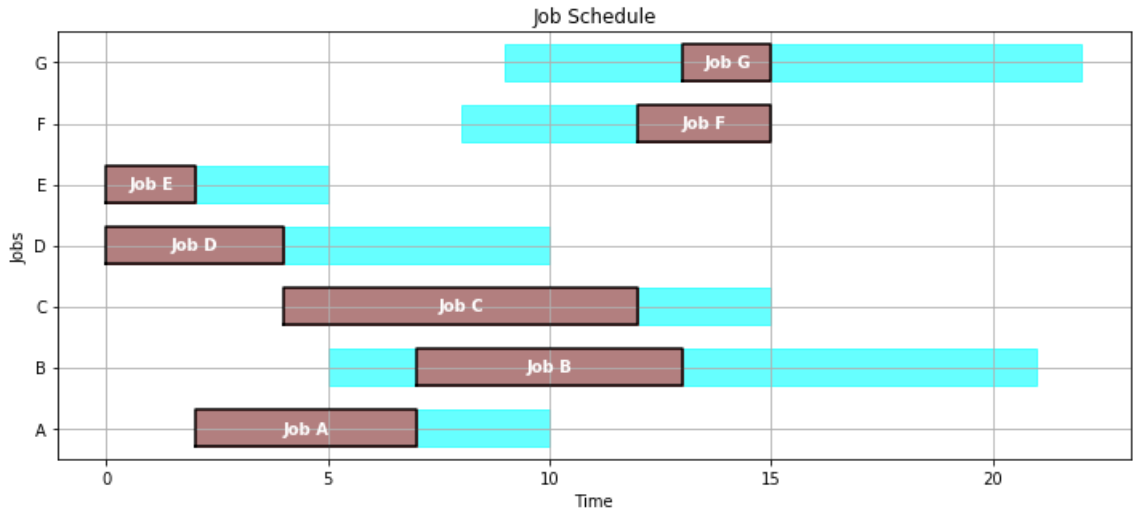
```
from pyomo.environ import *
from IPython.display import display
import pandas as pd

MACHINES = ['A','B']

def schedule_machines(JOBS, MACHINES):

    # create model
    m = ConcreteModel()

    # index set to simplify notation
    m.J = Set(initialize=JOBS.keys())
    m.M = Set(initialize=MACHINES)
    m.PAIRS = Set(initialize = m.J * m.J, dimen=2, filter=lambda m, j, k : j < k)

    # decision variables
    m.start      = Var(m.J, bounds=(0, 1000))
    m.makespan   = Var(domain=NonNegativeReals)
    m.pastdue    = Var(m.J, domain=NonNegativeReals)
    m.early      = Var(m.J, domain=NonNegativeReals)

    # additional decision variables for use in the objecive
    m.ispastdue  = Var(m.J, domain=Binary)
    m.maxpastdue = Var(domain=NonNegativeReals)

    # for binary assignment of jobs to machines
    m.z = Var(m.J, m.M, domain=Binary)

    # for modeling disjunctive constraints
    m.y = Var(m.PAIRS, domain=Binary)
    BigM = max([JOBS[j]['release'] for j in m.J]) + sum([JOBS[j]['duration'] for j in m.
J])

    m.OBJ = Objective(expr = sum(m.pastdue[j] for j in m.J) + m.makespan - sum(m.early[j
] for j in m.J), sense = minimize)

    m.c1 = Constraint(m.J, rule=lambda m, j:
            m.start[j] >= JOBS[j]['release'])
    m.c2 = Constraint(m.J, rule=lambda m, j:
            m.start[j] + JOBS[j]['duration'] + m.early[j] == JOBS[j]['due'] + m.pastdue[
j])
    m.c3 = Constraint(m.J, rule=lambda m, j:
            sum(m.z[j,mach] for mach in m.M) == 1)
    m.c4 = Constraint(m.J, rule=lambda m, j:
            m.pastdue[j] <= BigM*m.ispastdue[j])
    m.c5 = Constraint(m.J, rule=lambda m, j:
            m.pastdue[j] <= m.maxpastdue)
```

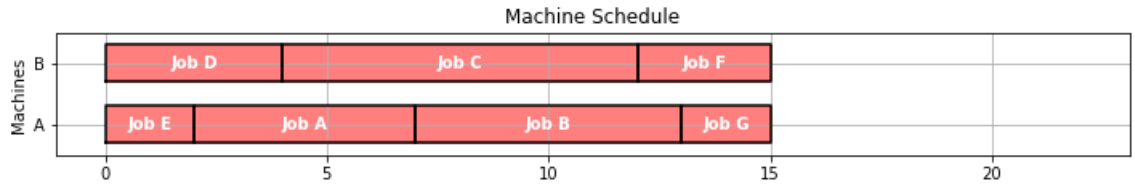
```
            m.pastdue[j] <= m.maxpastdue)
    m.c6 = Constraint(m.J, rule=lambda m, j:
            m.start[j] + JOBS[j]['duration'] <= m.makespan)
    m.d1 = Constraint(m.M, m.PAIRS, rule = lambda m, mach, j, k:
            m.start[j] + JOBS[j]['duration'] <= m.start[k] + BigM*(m.y[j,k] + (1-m.z[j,ma
ch]) + (1-m.z[k,mach])))
    m.d2 = Constraint(m.M, m.PAIRS, rule = lambda m, mach, j, k:
            m.start[k] + JOBS[k]['duration'] <= m.start[j] + BigM*((1-m.y[j,k]) + (1-m.z
j,mach]) + (1-m.z[k,mach])))

    SolverFactory('cbc').solve(m).write()

    SCHEDULE = {}
    for j in m.J:
        SCHEDULE[j] = {
            'start': m.start[j](),
            'finish': m.start[j]() + JOBS[j]['duration'],
            'machine': [mach for mach in MACHINES if m.z[j,mach]][0]
        }

    return SCHEDULE

SCHEDULE = schedule_machines(JOBS,MACHINES)
gantt(JOBS, SCHEDULE)
kpi(JOBS, SCHEDULE)
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: -15.0
  Upper bound: -15.0
  Number of objectives: 1
  Number of constraints: 127
  Number of variables: 66
  Number of nonzeros: 505
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  User time: -1.0
  Termination condition: optimal
  Error rc: 0
  Time: 0.5222530364990234
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

Out[12]:

```
{'Fraction on Time': 1.0,
 'Makespan': 15.0,
 'Max Pastdue': 0,
 'Number Pastdue': 0,
 'Number on Time': 7,
 'Sum of Pastdue': 0}
```


Job Schedule
G
F
E
D
C
B
A
Jobs
Job G
Job F
Job E
Job D
Job C
Job B
Job A
0
5
10
15
20
Time


Machine Schedule
B
A
Machines
Job D
Job C
Job F
Job E
Job A
Job B
Job G
0
5
10
15
20

In [ ]:

4.2 Job Shop Scheduling

In [4]:

```
!pip install -q pyomo
!apt-get install -y -qq glpk-utils
!apt-get install -y -qq coinor-cbc
```

```
/bin/sh: apt-get: command not found
/bin/sh: apt-get: command not found
```

In [ ]:

```
from pyomo.environ import *
from pyomo.gdp import *

#solver = SolverFactory('glpk')
solver = SolverFactory('cbc', executable='/usr/bin/cbc')
#solver = SolverFactory('gurobi', executable='/usr/local/bin/gurobi.sh')
```

Background

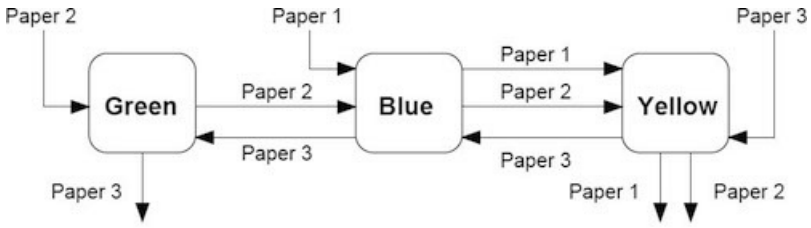
A job shop consists of a set of distinct machines that process jobs. Each job is a series of tasks that require use of particular machines for known durations, and which must be completed in specified order. The job shop scheduling problem is to schedule the jobs on the machines to minimize the time necessary to process all jobs (i.e, the makespan) or some other metric of productivity. Job shop scheduling is one of the classic problems in Operations Research.

Data consists of two tables. The first table is decomposition of the jobs into a series of tasks. Each task lists a job name, name of the required machine, and task duration. The second table list task pairs where the first task must be completed before the second task can be started. This formulation is quite general, but can also specify situations with no feasible solutions.

Job Shop Example

The following example of a job shop is from from Christelle Gueret, Christian Prins, Marc Sevaux, "Applications of Optimization with Xpress-MP," Dash Optimization, 2000.

In this example, there are three printed paper products that must pass through color printing presses in a particular order. The given data consists of a flowsheet showing the order in which each job passes through the color presses



and a table of data showing, in minutes, the amount of time each job requires on each machine.

Machine	Color	Paper 1	Paper 2	Paper 3
1	Blue	45	20	12
2	Green	-	10	17
3	Yellow	10	34	28

What is the minimum amount of time (i.e, what is the makespan) for this set of jobs?

Task Decomposition

The first step in the analysis is to decompose the process into a series of tasks. Each task is a (job,machine) pair. Some tasks cannot start until a prerequisite task is completed.

Task (Job,Machine)	Duration	Prerequisite Task
(Paper 1, Blue)	45	-
(Paper 1, Yellow)	10	(Paper 1,Blue)
(Paper 2, Blue)	20	(Paper 2, Green)
(Paper 2, Green)	10	-
(Paper 2, Yellow)	34	(Paper 2, Blue)
(Paper 3, Blue)	12	(Paper 3, Yellow)
(Paper 3, Green)	17	(Paper 3, Blue)
(Paper 3, Yellow)	28	-

We convert this to a JSON style representation where tasks are denoted by (Job,Machine) tuples in Python. The task data is stored in a Python dictionary indexed by (Job,Machine) tuples. The task data conists of a dictionary with duration ('dur') and (Job,Machine) pair for any prerequisite task.

In [1]:

```
TASKS = {
    ('Paper_1','Blue')   : {'dur': 45, 'prec': None},
    ('Paper_1','Yellow') : {'dur': 10, 'prec': ('Paper_1','Blue')},
    ('Paper_2','Blue')   : {'dur': 20, 'prec': ('Paper_2','Green')},
    ('Paper_2','Green')  : {'dur': 10, 'prec': None},
    ('Paper_2','Yellow') : {'dur': 34, 'prec': ('Paper_2','Blue')},
    ('Paper_3','Blue')   : {'dur': 12, 'prec': ('Paper_3','Yellow')},
    ('Paper_3','Green')  : {'dur': 17, 'prec': ('Paper_3','Blue')},
    ('Paper_3','Yellow') : {'dur': 28, 'prec': None},
}
```

Model Formulation

Each task is represented as an ordered pair (j, m) where j is a job, and m is a machine.

Parameter	Description
$\text{dur}_{j,m}$	Duration of task (j, m)
$\text{prec}_{j,m}$	A task (k, n) that must be completed before task (j, m) $= \text{Prec}_{j,m}$

Decision Variables	Description
makespan	Completion of all jobs
$\text{start}_{j,m}$	Start time for task (j, m)
$y_{j,k,m}$	boolean variable for tasks (i, m) and (j, m) on machine m where $j < k$

Upper and lower bounds on the start and completion of task (j, m)

$$\begin{aligned} \text{start}_{j,m} &\geq 0 \\ \text{start}_{j,m} + \text{Dur}_{j,m} &\leq \text{makespan} \end{aligned}$$

Satisfying prerequisite tasks

$$\text{start}_{k,n} + \text{Dur}_{k,n} \leq \text{start}_{j,m} \quad \text{for } (k, n) = \text{Prec}_{j,m}$$

Disjunctive Constraints

If M is big enough, then satisfying

$$\begin{aligned} \text{start}_{j,m} + \text{Dur}_{j,m} &\leq \text{start}_{k,m} + M(1 - y_{j,k,m}) \\ \text{start}_{k,m} + \text{Dur}_{k,m} &\leq \text{start}_{j,m} + My_{j,k,m} \end{aligned}$$

avoids conflicts for use of the same machine.

Pyomo Implementation

The job shop scheduling problem is implemented below in Pyomo. The implementation consists of of a function JobShop(TASKS) that accepts a dictionary of tanks and returns a pandas dataframe containing an optimal schedule of tasks. An optional argument to JobShop allows one to specify a solver.

In [16]:

```
def JobShop(TASKS):

    model = ConcreteModel()

    model.TASKS     = Set(initialize=TASKS.keys(), dimen=2)
    model.JOBS      = Set(initialize=set([j for (j,m) in TASKS.keys()]))
    model.MACHINES = Set(initialize=set([m for (j,m) in TASKS.keys()]))
    model.TASKORDER = Set(initialize = model.TASKS * model.TASKS, dimen=4,
        filter = lambda model,j,m,k,n: (k,n) == TASKS[(j,m)]['prec'])
    model.DISJUNCTIONS = Set(initialize=model.JOBS * model.JOBS * model.MACHINES, dimen
=3,
        filter = lambda model,j,k,m: j < k and (j,m) in model.TASKS and (k,m) in model.T
ASKS)

    t_max = sum([TASKS[(j,m)]['dur'] for (j,m) in TASKS.keys()])

    model.makespan = Var(bounds=(0, t_max))
    model.start = Var(model.TASKS, bounds=(0, t_max))

    model.obj = Objective(expr = model.makespan, sense = minimize)

    model.fini = Constraint(model.TASKS, rule=lambda model,j,m:
        model.start[j,m] + TASKS[(j,m)]['dur'] <= model.makespan)

    model.prec = Constraint(model.TASKORDER, rule=lambda model,j,m,k,n:
        model.start[k,n] + TASKS[(k,n)]['dur'] <= model.start[j,m])

    model.disj = Disjunction(model.DISJUNCTIONS, rule=lambda model,j,k,m:
        [model.start[j,m] + TASKS[(j,m)]['dur'] <= model.start[k,m],
         model.start[k,m] + TASKS[(k,m)]['dur'] <= model.start[j,m]])

    TransformationFactory('gdp.chull').apply_to(model)
    solver.solve(model)

    results = [{'Job': j,
                'Machine': m,
                'Start': model.start[j, m](),
                'Duration': TASKS[(j, m)]['dur'],
                'Finish': model.start[j, m]() + TASKS[(j, m)]['dur']}
               for j,m in model.TASKS]
    return results

results = JobShop(TASKS)
results
```

Out[16]:

```
[{'Duration': 1.5,
  'Finish': 25.5,
  'Job': 'C1',
  'Machine': 'Packaging',
  'Start': 24.0},
 {'Duration': 1.5,
  'Finish': 24.0,
  'Job': 'A2',
  'Machine': 'Packaging',
  'Start': 22.5},
 {'Duration': 4,
  'Finish': 18.5,
  'Job': 'A1',
  'Machine': 'Separator',
  'Start': 14.5},
 {'Duration': 5,
  'Finish': 5.0,
  'Job': 'C1',
  'Machine': 'Separator',
  'Start': 0.0},
```

```
{'Duration': 1,
 'Finish': 28.0,
 'Job': 'B2',
 'Machine': 'Packaging',
 'Start': 27.0},
{'Duration': 1.5,
 'Finish': 19.0,
 'Job': 'C2',
 'Machine': 'Packaging',
 'Start': 17.5},
{'Duration': 3,
 'Finish': 17.5,
 'Job': 'C2',
 'Machine': 'Reactor',
 'Start': 14.5},
{'Duration': 1, 'Finish': 1.0, 'Job': 'A1', 'Machine': 'Mixer', 'Start': 0.0},
{'Duration': 1, 'Finish': 2.0, 'Job': 'A2', 'Machine': 'Mixer', 'Start': 1.0},
{'Duration': 5,
 'Finish': 11.0,
 'Job': 'A2',
 'Machine': 'Reactor',
 'Start': 6.0},
{'Duration': 1,
 'Finish': 21.5,
 'Job': 'B1',
 'Machine': 'Packaging',
 'Start': 20.5},
{'Duration': 3,
 'Finish': 20.5,
 'Job': 'C1',
 'Machine': 'Reactor',
 'Start': 17.5},
{'Duration': 5,
 'Finish': 14.5,
 'Job': 'C2',
 'Machine': 'Separator',
 'Start': 9.5},
{'Duration': 4.5,
 'Finish': 27.0,
 'Job': 'B2',
 'Machine': 'Separator',
 'Start': 22.5},
{'Duration': 1.5,
 'Finish': 20.5,
 'Job': 'A1',
 'Machine': 'Packaging',
 'Start': 19.0},
{'Duration': 5,
 'Finish': 6.0,
 'Job': 'A1',
 'Machine': 'Reactor',
 'Start': 1.0},
{'Duration': 4,
 'Finish': 22.5,
 'Job': 'A2',
 'Machine': 'Separator',
 'Start': 18.5},
{'Duration': 4.5,
 'Finish': 9.5,
 'Job': 'B1',
 'Machine': 'Separator',
 'Start': 5.0}]
```


Printing Schedules

In [17]:

```
import pandas as pd

schedule = pd.DataFrame(results)

print('\nSchedule by Job')
print(schedule.sort_values(by=['Job','Start']).set_index(['Job', 'Machine']))

print('\nSchedule by Machine')
print(schedule.sort_values(by=['Machine','Start']).set_index(['Machine', 'Job']))
```

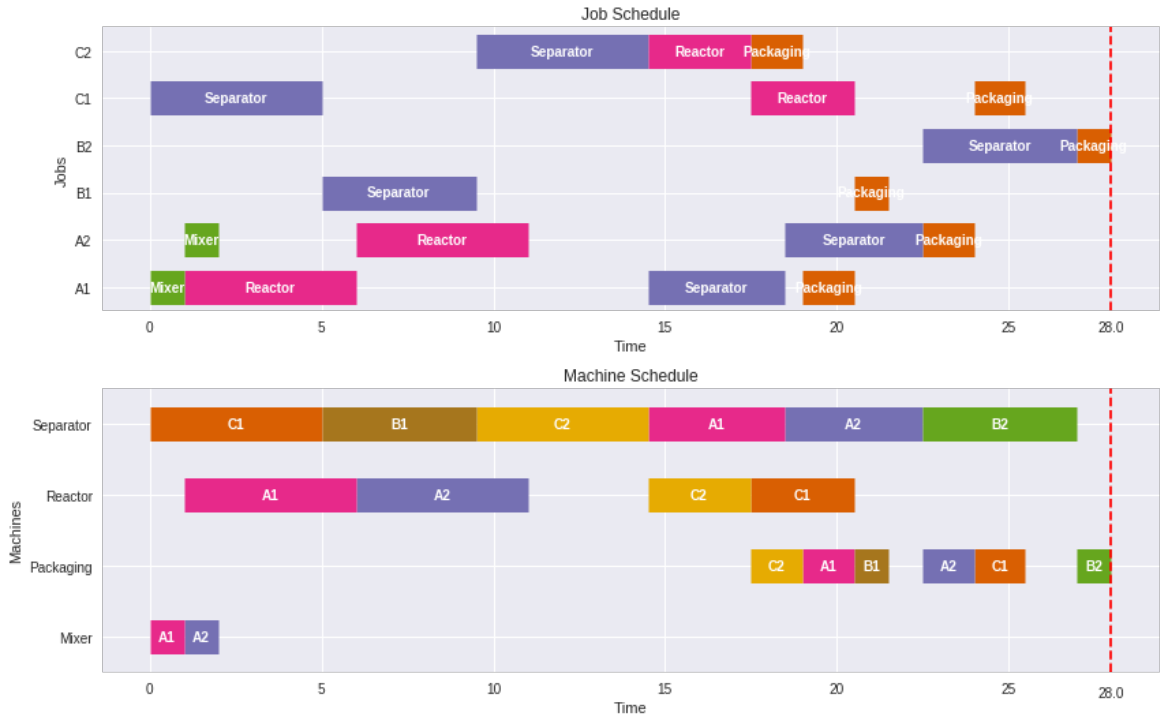
```
Schedule by Job
                Duration  Finish  Start
Job Machine
A1  Mixer            1.0     1.0    0.0
    Reactor          5.0     6.0    1.0
    Separator        4.0    18.5   14.5
    Packaging        1.5    20.5   19.0
A2  Mixer            1.0     2.0    1.0
    Reactor          5.0    11.0    6.0
    Separator        4.0    22.5   18.5
    Packaging        1.5    24.0   22.5
B1  Separator        4.5     9.5    5.0
    Packaging        1.0    21.5   20.5
B2  Separator        4.5    27.0   22.5
    Packaging        1.0    28.0   27.0
C1  Separator        5.0     5.0    0.0
    Reactor          3.0    20.5   17.5
    Packaging        1.5    25.5   24.0
C2  Separator        5.0    14.5    9.5
    Reactor          3.0    17.5   14.5
    Packaging        1.5    19.0   17.5

Schedule by Machine
                Duration  Finish  Start
Machine   Job
Mixer     A1         1.0     1.0    0.0
          A2         1.0     2.0    1.0
Packaging C2         1.5    19.0   17.5
          A1         1.5    20.5   19.0
          B1         1.0    21.5   20.5
          A2         1.5    24.0   22.5
          C1         1.5    25.5   24.0
          B2         1.0    28.0   27.0
Reactor   A1         5.0     6.0    1.0
          A2         5.0    11.0    6.0
          C2         3.0    17.5   14.5
          C1         3.0    20.5   17.5
Separator C1         5.0     5.0    0.0
          B1         4.5     9.5    5.0
          C2         5.0    14.5    9.5
          A1         4.0    18.5   14.5
          A2         4.0    22.5   18.5
          B2         4.5    27.0   22.5
```

Visualizing Results with Gantt Charts

In [18]:

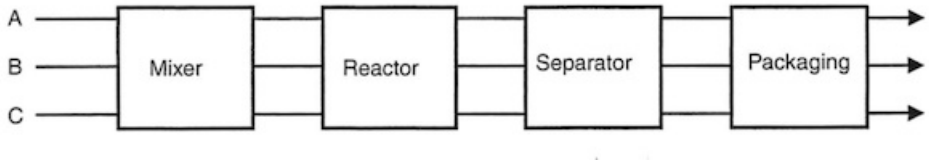
```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd

def Visualize(results):
```

```
    schedule = pd.DataFrame(results)
    JOBS = list(schedule['Job'].unique())
    MACHINES = list(schedule['Machine'].unique())
    makespan = schedule['Finish'].max()

    schedule.sort_values(by=['Job','Start'])
    schedule.set_index(['Job', 'Machine'], inplace=True)

    plt.figure(figsize=(12, 5 + (len(JOBS)+len(MACHINES))/4))
    plt.subplot(2,1,1)

    jdx = 0
    for j in sorted(JOBS):
        jdx += 1
        mdx = 0
        for m in MACHINES:
            mdx += 1
            c = mpl.cm.Dark2.colors[mdx%7]
            if (j,m) in schedule.index:
                plt.plot([schedule.loc[(j,m),'Start'],schedule.loc[(j,m),'Finish']],
                     [jdx,jdx],color = c,alpha=1.0,lw=25,solid_capstyle='butt')
                plt.text((schedule.loc[(j,m),'Start'] +
schedule.loc[(j,m),'Finish'])/2.0,jdx,
                    m, color='white', weight='bold',
                    horizontalalignment='center', verticalalignment='center')

    plt.ylim(0.5,jdx+0.5)
    plt.title('Job Schedule')
    plt.gca().set_yticks(range(1,1+len(JOBS)))
    plt.gca().set_yticklabels(sorted(JOBS))
    plt.plot([makespan,makespan],plt.ylim(),'r--')
    plt.text(makespan,plt.ylim()[0]-0.2,str(round(makespan,2)),
            horizontalalignment='center', verticalalignment='top')
    plt.xlabel('Time')
    plt.ylabel('Jobs')

    plt.subplot(2,1,2)
    mdx = 0
    for m in sorted(MACHINES):
        mdx += 1
        jdx = 0
        for j in JOBS:
            jdx += 1
            c = mpl.cm.Dark2.colors[jdx%7]
            if (j,m) in schedule.index:
                plt.plot([schedule.loc[(j,m),'Start'],schedule.loc[(j,m),'Finish']],
                     [mdx,mdx],color = c,alpha=1.0,lw=25,solid_capstyle='butt')
                plt.text((schedule.loc[(j,m),'Start'] +
schedule.loc[(j,m),'Finish'])/2.0,mdx,
                    j, color='white', weight='bold',
                    horizontalalignment='center', verticalalignment='center')

    plt.ylim(0.5,mdx+0.5)
    plt.title('Machine Schedule')
    plt.gca().set_yticks(range(1,1+len(MACHINES)))
    plt.gca().set_yticklabels(sorted(MACHINES))
    plt.plot([makespan,makespan],plt.ylim(),'r--')
    plt.text(makespan,plt.ylim()[0]-0.2,str(round(makespan,2)),
            horizontalalignment='center', verticalalignment='top')
    plt.xlabel('Time')
    plt.ylabel('Machines')

    plt.tight_layout()

Visualize(results)
```



Application to Scheduling of Batch Processes

We will now turn our attention to the application of the job shop scheduling problem to the short term scheduling of batch processes. We illustrate these techniques using Example II from Dunn (2013).



Process	Mixer	Reactor	Separator	Packaging
A	1.0	5.0	4.0	1.5
B	-	-	4.5	1.0
C	-	3.0	5.0	1.5

Single Product Strategies

Before going further, we create a function to streamline the generation of the TASKS dictionary.

In [2]:

```
def Recipe(jobs,machines,durations):
    TASKS = {}
    for j in jobs:
        prec = (None,None)
        for m,d in zip(machines,durations):
            task = (j,m)
            if prec == (None,None):
                TASKS.update({(j,m): {'dur': d, 'prec': None}})
            else:
                TASKS.update({(j,m): {'dur': d, 'prec': prec}})
            prec = task
    return TASKS

RecipeA = Recipe('A',['Mixer','Reactor','Separator','Packaging'],[1,5,4,1.5])
RecipeB = Recipe('B',['Separator','Packaging'],[4.5,1])
RecipeC = Recipe('C',['Separator','Reactor','Packaging'],[5,3,1.5])

Visualize(JobShop(RecipeA))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-184d66ea5167> in <module>
     16 RecipeC = Recipe('C',['Separator','Reactor','Packaging'],[5,3,1.5])
     17
---> 18 Visualize(JobShop(RecipeA))

NameError: name 'Visualize' is not defined
```

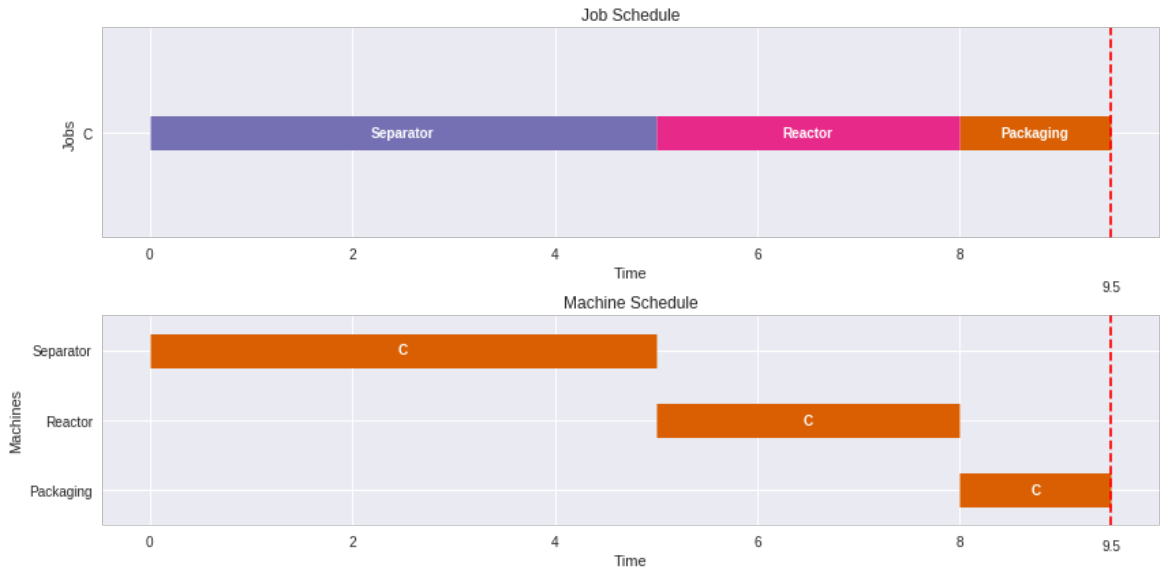
In [3]:

```
Visualize(JobShop(RecipeB))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-0b43ff55198c> in <module>
----> 1 Visualize(JobShop(RecipeB))

NameError: name 'Visualize' is not defined
```

In [21]:

```
Visualize(JobShop(RecipeC))
```



Overlapping Tasks

Let's now consider an optimal scheduling problem where we are wish to make two batches of Product A.

In [22]:

```
TASKS = Recipe(['A1','A2'],['Mixer','Reactor','Separator','Packaging'],[1,5,4,1.5])
results = JobShop(TASKS)
Visualize(results)
print("Makespan =", max([task['Finish'] for task in results]))
```

```
Makespan = 16.5
```

Earlier we found it tood 11.5 hours to produce one batch of product A. As we see here, we can produce a second batch with only 5.0 additional hours because some of the tasks overlap. The overlapping of tasks is the key to gaining efficiency in batch processing facilities.

Let's next consider production of a single batch each of products A, B, and C.

In [23]:

```
TASKS = RecipeA
TASKS.update(RecipeB)
TASKS.update(RecipeC)

results = JobShop(TASKS)
Visualize(results)
print("Makespan =", max([task['Finish'] for task in results]))
```

```
Makespan = 15.0
```

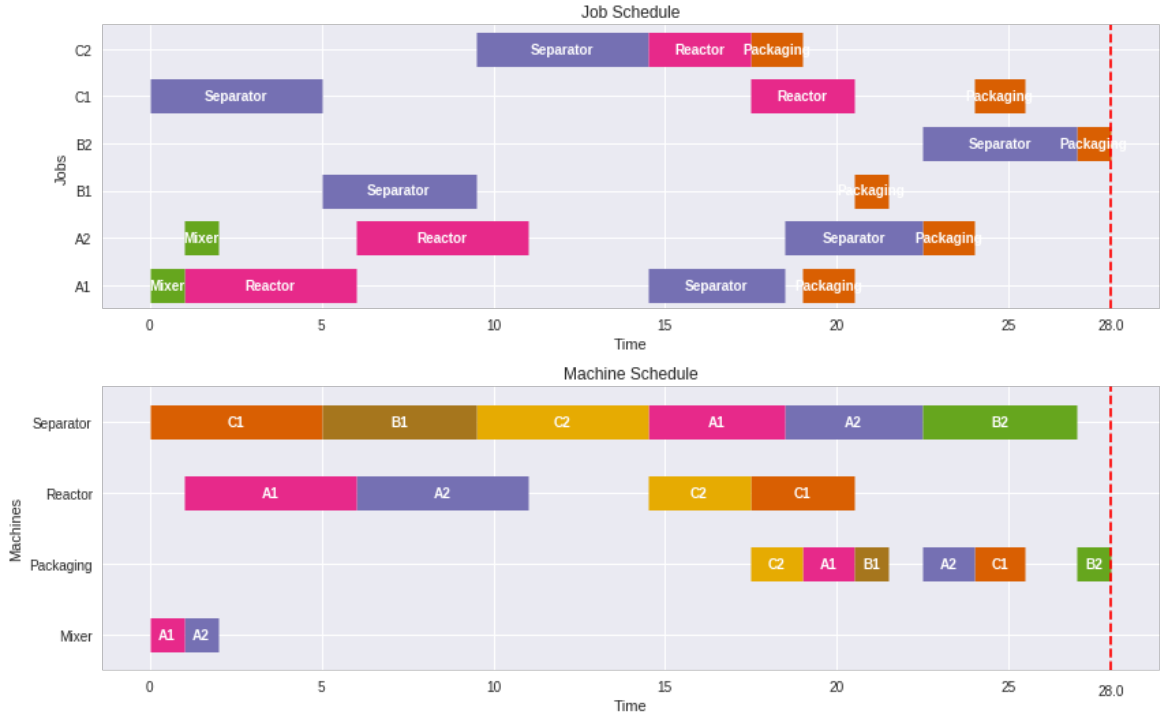
The individual production of A, B, and C required 11.5, 5.5, and 9.5 hours, respectively, for a total of 25.5 hours. As we see here, by scheduling the production simultaneously, we can get all three batches done in just 15 hours.

As we see below, each additional set of three products takes an additionl 13 hours. So there is considerable efficiency gained by scheduling over longer intervals whenever possible.

In [24]:

```
TASKS = Recipe(['A1','A2'],['Mixer','Reactor','Separator','Packaging'],[1,5,4,1.5])
TASKS.update(Recipe(['B1','B2'],['Separator','Packaging'],[4.5,1]))
TASKS.update(Recipe(['C1','C2'],['Separator','Reactor','Packaging'],[5,3,1.5]))

results = JobShop(TASKS)
Visualize(results)
print("Makespan =", max([task['Finish'] for task in results]))
```

```
Makespan = 28.0
```



Unit Cleanout

A common feature in batch unit operations is a requirement that equipment be cleaned prior to reuse.

In most cases the time needed for clean out would be equipment and product specific. Bur for the purposes of illustration, we implement this policy with a single non-negative parameter $t_{clean} \geq 0$ which, if specified, requires a period no less than t_{clean} between the finish of one task and the start of another on every piece of equipment.

This is implemented by modifying the usual disjunctive constraints to avoid machine conflicts, i.e.,

$$\begin{aligned} \text{start}_{j,m} + \text{Dur}_{j,m} &\leq \text{start}_{k,m} + M(1 - y_{j,k,m}) \\ \text{start}_{k,m} + \text{Dur}_{k,m} &\leq \text{start}_{j,m} + My_{j,k,m} \end{aligned}$$

to read

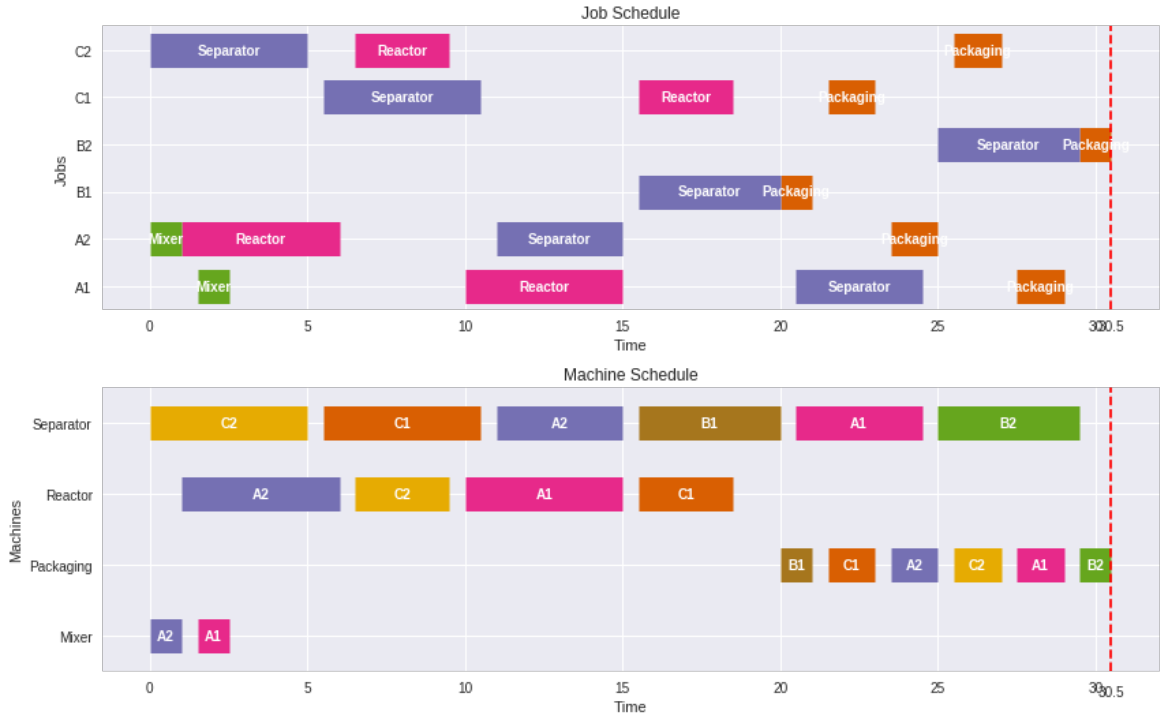
$$\begin{aligned} \text{start}_{j,m} + \text{Dur}_{j,m} + t_{clean} &\leq \text{start}_{k,m} + M(1 - y_{j,k,m}) \\ \text{start}_{k,m} + \text{Dur}_{k,m} + t_{clean} &\leq \text{start}_{j,m} + My_{j,k,m} \end{aligned}$$

for sufficiently large M.

In [27]:

```
def JobShop(TASKS, tclean=0):

    model = ConcreteModel()

    model.TASKS     = Set(initialize=TASKS.keys(), dimen=2)
    model.JOBS      = Set(initialize=set([j for (j,m) in TASKS.keys()]))
    model.MACHINES = Set(initialize=set([m for (j,m) in TASKS.keys()]))
    model.TASKORDER = Set(initialize = model.TASKS * model.TASKS, dimen=4,
        filter = lambda model,j,m,k,n: (k,n) == TASKS[(j,m)]['prec'])
    model.DISJUNCTIONS = Set(initialize=model.JOBS * model.JOBS * model.MACHINES, dimen
=3,
        filter = lambda model,j,k,m: j < k and (j,m) in model.TASKS and (k,m) in model.T
ASKS)

    t_max = sum([TASKS[(j,m)]['dur'] for (j,m) in TASKS.keys()])

    model.makespan = Var(bounds=(0, t_max))
    model.start = Var(model.TASKS, bounds=(0, t_max))

    model.obj = Objective(expr = model.makespan, sense = minimize)

    model.fini = Constraint(model.TASKS, rule=lambda model,j,m:
        model.start[j,m] + TASKS[(j,m)]['dur'] <= model.makespan)

    model.prec = Constraint(model.TASKORDER, rule=lambda model,j,m,k,n:
        model.start[k,n] + TASKS[(k,n)]['dur'] <= model.start[j,m])

    model.disj = Disjunction(model.DISJUNCTIONS, rule=lambda model,j,k,m:
        [model.start[j,m] + TASKS[(j,m)]['dur'] + tclean <= model.start[k,m],
         model.start[k,m] + TASKS[(k,m)]['dur'] + tclean <= model.start[j,m]])

    TransformationFactory('gdp.chull').apply_to(model)
    solver.solve(model)

    results = [{'Job': j,
                'Machine': m,
                'Start': model.start[j, m](),
                'Duration': TASKS[(j, m)]['dur'],
                'Finish': model.start[j, m]() + TASKS[(j, m)]['dur']}
               for j,m in model.TASKS]
    return results

results = JobShop(TASKS, tclean=0.5)
Visualize(results)
print("Makespan =", max([task['Finish'] for task in results]))
```

```
Makespan = 30.5
```



Zero Wait Policy

One of the issues in the use of job shop scheduling for batch processing are situations where there it isn't possible to store intermediate materials. If there is no way to store intermediates, either in the processing equipment or in external vessels, then a **zero-wait** policy may be appropriate.

A zero-wait policy requires subsequent processing machines to be available immediately upon completion of any task. To implement this policy, the usual precident sequencing constraint of a job shop scheduling problem, i.e.,

$$\text{start}_{k,n} + \text{Dur}_{k,n} \leq \text{start}_{j,m} \quad \text{for } (k,n) = \text{Prec}_{j,m}$$

is changed to

$$\text{start}_{k,n} + \text{Dur}_{k,n} = \text{start}_{j,m} \quad \text{for } (k,n) = \text{Prec}_{j,m} \text{ and ZW is True}$$

if the zero-wait policy is in effect.

While this could be implemented on an equipment or product specific basis, here we add an optional ZW flag to the JobShop function that, by default, is set to False.

In [29]:

```
def JobShop(TASKS, tclean=0, ZW=False):

    model = ConcreteModel()

    model.TASKS     = Set(initialize=TASKS.keys(), dimen=2)
    model.JOBS      = Set(initialize=set([j for (j,m) in TASKS.keys()]))
    model.MACHINES = Set(initialize=set([m for (j,m) in TASKS.keys()]))
    model.TASKORDER = Set(initialize = model.TASKS * model.TASKS, dimen=4,
        filter = lambda model,j,m,k,n: (k,n) == TASKS[(j,m)]['prec'])
    model.DISJUNCTIONS = Set(initialize=model.JOBS * model.JOBS * model.MACHINES, dimen
=3,
        filter = lambda model,j,k,m: j < k and (j,m) in model.TASKS and (k,m) in model.T
ASKS)

    t_max = sum([TASKS[(j,m)]['dur'] for (j,m) in TASKS.keys()])

    model.makespan = Var(bounds=(0, t_max))
    model.start = Var(model.TASKS, bounds=(0, t_max))

    model.obj = Objective(expr = model.makespan, sense = minimize)

    model.fini = Constraint(model.TASKS, rule=lambda model,j,m:
        model.start[j,m] + TASKS[(j,m)]['dur'] <= model.makespan)

    if ZW:
        model.prec = Constraint(model.TASKORDER, rule=lambda model,j,m,k,n:
            model.start[k,n] + TASKS[(k,n)]['dur'] == model.start[j,m])
    else:
        model.prec = Constraint(model.TASKORDER, rule=lambda model,j,m,k,n:
            model.start[k,n] + TASKS[(k,n)]['dur'] <= model.start[j,m])

    model.disj = Disjunction(model.DISJUNCTIONS, rule=lambda model,j,k,m:
        [model.start[j,m] + TASKS[(j,m)]['dur'] + tclean <= model.start[k,m],
         model.start[k,m] + TASKS[(k,m)]['dur'] + tclean <= model.start[j,m]])

    TransformationFactory('gdp.chull').apply_to(model)
    solver.solve(model)

    results = [{'Job': j,
                'Machine': m,
                'Start': model.start[j, m](),
                'Duration': TASKS[(j, m)]['dur'],
                'Finish': model.start[j, m]() + TASKS[(j, m)]['dur']}
               for j,m in model.TASKS]
    return results

results = JobShop(TASKS, tclean=0.5, ZW=True)
Visualize(results)
print("Makespan =", max([task['Finish'] for task in results]))
```

```
Makespan = 32.0
```

Benchmark Problems LA19

The file `jobshop1.txt` is a well known collection of 82 benchmark problems is a well-known collection of job shop scheduling problems in the OR-Library maintained by J. E. Beasley. The data format for each example consists of a single line for each job. The data on each line is a sequence of (machine number, time) pairs showing the order in which machines process each job.

LA19 is a benchmark problem for job shop scheduling introduced by Lawrence in 1984, and a solution presented by Cook and Applegate in 1991. The following cell may take many minutes to hours to run, depending on the choice of solver and hardware.

In [30]:

```
data = """
2  44  3   5  5  58  4  97  0   9  7  84  8  77  9  96  1  58  6  89
4  15  7  31  1  87  8  57  0  77  3  85  2  81  5  39  9  73  6  21
9  82  6  22  4  10  3  70  1  49  0  40  8  34  2  48  7  80  5  71
1  91  2  17  7  62  5  75  8  47  4  11  3   7  6  72  9  35  0  55
6  71  1  90  3  75  0  64  2  94  8  15  4  12  7  67  9  20  5  50
7  70  5  93  8  77  2  29  4  58  6  93  3  68  1  57  9   7  0  52
6  87  1  63  4  26  5   6  2  82  3  27  7  56  8  48  9  36  0  95
0  36  5  15  8  41  9  78  3  76  6  84  4  30  7  76  2  36  1   8
5  88  2  81  3  13  6  82  4  54  7  13  8  29  9  40  1  78  0  75
9  88  4  54  6  64  7  32  0  52  2   6  8  54  5  82  3   6  1  26
"""

TASKS = {}
prec = ''

lines = data.splitlines()
job= 0
for line in lines[1:]:
    j = "J{0:1d}".format(job)
    nums = line.split()
    prec = ''
    for m,dur in zip(nums[::2],nums[1::2]):
        task = (j,'M{0:s}'.format(m))
        if prec:
            TASKS[task] = {'dur':int(dur), 'prec':prec}
        else:
            TASKS[task] = {'dur':int(dur), 'prec':None}
        prec = task
    job += 1

Visualize(JobShop(TASKS))
```

```
  Signal handler called from  /usr/lib/python3.6/subprocess.py _try_wait 1404
  Waiting...
  Signaled process 319 with signal 2
ERROR: Solver (cbc) returned non-zero return code (-1)
ERROR: Solver log: Welcome to the CBC MILP Solver Version: 2.9.9 Build Date:
    Aug 21 2017

    command line - /usr/bin/cbc -printingOptions all -import
    /content/tmp_4xbvphs.pyomo.lp -stat=1 -solve -solu
    /content/tmp_4xbvphs.pyomo.soln (default strategy 1) Option for
    printingOptions changed from normal to all Presolve 2890 (-1351) rows,
    1451 (-1351) columns and 8480 (-1801) elements Statistics for presolved
    model Original problem has 900 integers (900 of which binary) Presolved
    problem has 450 integers (450 of which binary) ==== 1450 zero objective 2
    different 1450 variables have objective of 0 1 variables have objective of
    1 ==== absolute objective values 2 different 1450 variables have objective
    of 0 1 variables have objective of 1 ==== for integers 450 zero objective
    1 different 450 variables have objective of 0 ==== for integers absolute
    objective values 1 different 450 variables have objective of 0 ===== end
    objective counts


    Problem has 2890 rows, 1451 columns (1 with objective) and 8480 elements
    Column breakdown: 0 of type 0.0->inf, 1001 of type 0.0->up, 0 of type
    lo->inf, 0 of type lo->up, 0 of type free, 0 of type fixed, 0 of type
    -inf->0.0, 0 of type -inf->up, 450 of type 0.0->1.0 Row breakdown: 0 of
    type E 0.0, 0 of type E 1.0, 0 of type E -1.0, 0 of type E other, 0 of
    type G 0.0, 0 of type G 1.0, 0 of type G other, 1350 of type L 0.0, 0 of
    type L 1.0, 1540 of type L other, 0 of type Range 0.0->1.0, 0 of type
    Range other, 0 of type Free Continuous objective value is 617 - 0.02
    seconds Cgl0003I 0 fixed, 0 tightened bounds, 900 strengthened rows, 0
    substitutions Cgl0003I 0 fixed, 0 tightened bounds, 48 strengthened rows,
    0 substitutions Cgl0004I processed model has 2890 rows, 1451 columns (450
```

```
integer (450 of which binary)) and 8480 elements Cbc0038I Initial state -
432 integers unsatisfied sum - 31.8732 Cbc0038I Pass   1: suminf.
0.08567 (1) obj. 4186 iterations 655 Cbc0038I Pass   2: suminf.    0.00000
(0) obj. 4186 iterations 2 Cbc0038I Solution found of 4186 Cbc0038I
Relaxing continuous gives 4186 Cbc0038I Before mini branch and bound, 18
integers at bound fixed and 478 continuous Cbc0038I Mini branch and bound
did not improve solution (0.18 seconds) Cbc0038I Round again with cutoff
of 3829.1 Cbc0038I Pass   3: suminf.    0.14188 (5) obj. 3829.1 iterations
4 Cbc0038I Pass   4: suminf.    0.00000 (0) obj. 3829.1 iterations 15
Cbc0038I Solution found of 3829.1 Cbc0038I Relaxing continuous gives 3762
Cbc0038I Before mini branch and bound, 18 integers at bound fixed and 474
continuous Cbc0038I Mini branch and bound did not improve solution (0.25
seconds) Cbc0038I Round again with cutoff of 3133 Cbc0038I Pass   5:
suminf.    0.27367 (11) obj. 3133 iterations 8 Cbc0038I Pass   6: suminf.
0.00000 (0) obj. 3133 iterations 32 Cbc0038I Solution found of 3133
Cbc0038I Relaxing continuous gives 3091 Cbc0038I Before mini branch and
bound, 18 integers at bound fixed and 468 continuous Cbc0038I Mini branch
and bound did not improve solution (0.33 seconds) Cbc0038I Round again
with cutoff of 2348.8 Cbc0038I Pass   7: suminf.    0.62110 (27) obj.
2348.8 iterations 21 Cbc0038I Pass   8: suminf.    0.02174 (4) obj. 2348.8
iterations 74 Cbc0038I Pass   9: suminf.    0.00724 (1) obj. 2348.8
iterations 20 Cbc0038I Pass  10: suminf.    0.00000 (0) obj. 2348.8
iterations 5 Cbc0038I Solution found of 2348.8 Cbc0038I Relaxing
continuous gives 2344 Cbc0038I Before mini branch and bound, 18 integers
at bound fixed and 454 continuous Cbc0038I Mini branch and bound did not
improve solution (0.42 seconds) Cbc0038I Round again with cutoff of 1825.9
Cbc0038I Pass  11: suminf.    1.07503 (43) obj. 1825.9 iterations 24
Cbc0038I Pass  12: suminf.    0.42644 (28) obj. 1825.9 iterations 38
Cbc0038I Pass  13: suminf.    0.04707 (11) obj. 1825.9 iterations 75
Cbc0038I Pass  14: suminf.    0.07513 (7) obj. 1825.9 iterations 70
Cbc0038I Pass  15: suminf.    0.02039 (5) obj. 1825.9 iterations 34
Cbc0038I Pass  16: suminf.    0.02302 (1) obj. 1825.9 iterations 18
Cbc0038I Pass  17: suminf.    0.00000 (0) obj. 1825.9 iterations 2
Cbc0038I Solution found of 1825.9 Cbc0038I Relaxing continuous gives 1809
Cbc0038I Before mini branch and bound, 18 integers at bound fixed and 437
continuous Cbc0038I Mini branch and bound did not improve solution (0.53
seconds) Cbc0038I Freeing continuous variables gives a solution of 1809
Cbc0038I Round again with cutoff of 1332.2 Cbc0038I Pass  18: suminf.
1.96379 (71) obj. 1332.2 iterations 52 Cbc0038I Pass  19: suminf.
0.58600 (47) obj. 1332.2 iterations 59 Cbc0038I Pass  20: suminf.
0.13103 (23) obj. 1332.2 iterations 104 Cbc0038I Pass  21: suminf.
0.15270 (16) obj. 1332.2 iterations 78 Cbc0038I Pass  22: suminf.
0.07938 (13) obj. 1332.2 iterations 65 Cbc0038I Pass  23: suminf.
0.13261 (12) obj. 1332.2 iterations 66 Cbc0038I Pass  24: suminf.
0.06156 (9) obj. 1332.2 iterations 42 Cbc0038I Pass  25: suminf.
0.14238 (11) obj. 1332.2 iterations 38 Cbc0038I Pass  26: suminf.
5.81772 (145) obj. 1332.2 iterations 323 Cbc0038I Pass  27: suminf.
3.38257 (103) obj. 1332.2 iterations 83 Cbc0038I Pass  28: suminf.
1.40182 (57) obj. 1332.2 iterations 103 Cbc0038I Pass  29: suminf.
0.32342 (30) obj. 1332.2 iterations 73 Cbc0038I Pass  30: suminf.
0.12258 (12) obj. 1332.2 iterations 126 Cbc0038I Pass  31: suminf.
0.04616 (7) obj. 1332.2 iterations 66 Cbc0038I Pass  32: suminf.
0.04705 (5) obj. 1332.2 iterations 35 Cbc0038I Pass  33: suminf.
0.03656 (5) obj. 1332.2 iterations 22 Cbc0038I Pass  34: suminf.
0.01994 (3) obj. 1332.2 iterations 24 Cbc0038I Pass  35: suminf.
0.01056 (4) obj. 1332.2 iterations 21 Cbc0038I Pass  36: suminf.
5.73919 (154) obj. 1332.2 iterations 361 Cbc0038I Pass  37: suminf.
3.51088 (113) obj. 1332.2 iterations 85 Cbc0038I Pass  38: suminf.
2.54396 (97) obj. 1332.2 iterations 51 Cbc0038I Pass  39: suminf.
1.68684 (66) obj. 1332.2 iterations 71 Cbc0038I Pass  40: suminf.
0.40749 (40) obj. 1332.2 iterations 99 Cbc0038I Pass  41: suminf.
0.15058 (23) obj. 1332.2 iterations 62 Cbc0038I Pass  42: suminf.
0.13852 (9) obj. 1332.2 iterations 117 Cbc0038I Pass  43: suminf.
0.04514 (7) obj. 1332.2 iterations 37 Cbc0038I Pass  44: suminf.
0.04947 (5) obj. 1332.2 iterations 31 Cbc0038I Pass  45: suminf.
0.03443 (5) obj. 1332.2 iterations 15 Cbc0038I Pass  46: suminf.
4.60897 (136) obj. 1332.2 iterations 352 Cbc0038I Pass  47: suminf.
2.93035 (111) obj. 1332.2 iterations 54 Cbc0038I No solution found this
major pass Cbc0038I Before mini branch and bound, 5 integers at bound
```

```
fixed and 183 continuous Cbc0038I Full problem 2890 rows 1451 columns,
reduced to 2697 rows 1263 columns - 15 fixed gives 2627, 1227 - still too
large Cbc0038I Mini branch and bound did not improve solution (0.97
seconds) Cbc0038I After 0.97 seconds - Feasibility pump exiting with
objective of 1809 - took 0.84 seconds Cbc0012I Integer solution of 1809
found by feasibility pump after 0 iterations and 0 nodes (0.97 seconds)
Cbc0031I 739 added rows had average density of 5.0460081 Cbc0013I At root
node, 739 cuts changed objective from 617 to 682.11199 in 21 passes
Cbc0014I Cut generator 0 (Probing) - 16701 row cuts average 5.1 elements,
0 column cuts (190 active)  in 0.184 seconds - new frequency is 1 Cbc0014I
Cut generator 1 (Gomory) - 2657 row cuts average 7.1 elements, 0 column
cuts (0 active)  in 0.149 seconds - new frequency is 1 Cbc0014I Cut
generator 2 (Knapsack) - 0 row cuts average 0.0 elements, 0 column cuts (0
active)  in 0.023 seconds - new frequency is -100 Cbc0014I Cut generator 3
(Clique) - 0 row cuts average 0.0 elements, 0 column cuts (0 active)  in
0.004 seconds - new frequency is -100 Cbc0014I Cut generator 4
(MixedIntegerRounding2) - 2000 row cuts average 2.2 elements, 0 column
cuts (0 active)  in 0.050 seconds - new frequency is 1 Cbc0014I Cut
generator 5 (FlowCover) - 139 row cuts average 2.0 elements, 0 column cuts
(0 active)  in 0.043 seconds - new frequency is -100 Cbc0014I Cut
generator 6 (TwoMirCuts) - 2174 row cuts average 4.1 elements, 0 column
cuts (0 active)  in 0.142 seconds - new frequency is 1 Cbc0010I After 0
nodes, 1 on tree, 1809 best solution, best possible 682.11199 (7.51
seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 2431
rows 1145 columns - 4 fixed gives 2419, 1137 - still too large Cbc0038I
Full problem 2890 rows 1451 columns, reduced to 2356 rows 1095 columns - 2
fixed gives 2350, 1091 - still too large Cbc0010I After 100 nodes, 59 on
tree, 1809 best solution, best possible 682.11199 (47.98 seconds) Cbc0038I
Full problem 2890 rows 1451 columns, reduced to 2224 rows 1007 columns - 1
fixed gives 2221, 1005 - still too large Cbc0010I After 200 nodes, 112 on
tree, 1809 best solution, best possible 682.11199 (54.88 seconds) Cbc0012I
Integer solution of 1095 found by rounding after 93314 iterations and 294
nodes (57.95 seconds) Cbc0038I Full problem 2890 rows 1451 columns,
reduced to 1554 rows 569 columns - 1 fixed gives 1551, 567 - still too
large Cbc0010I After 300 nodes, 154 on tree, 1095 best solution, best
possible 682.11199 (58.97 seconds) Cbc0016I Integer solution of 1061 found
by strong branching after 95344 iterations and 302 nodes (59.11 seconds)
Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1804 rows 735
columns - 1 fixed gives 1801, 733 - still too large Cbc0010I After 400
nodes, 201 on tree, 1061 best solution, best possible 682.11199 (74.64
seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1695
rows 662 columns - 1 fixed gives 1692, 660 - still too large Cbc0010I
After 500 nodes, 252 on tree, 1061 best solution, best possible 682.11199
(80.64 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to
1671 rows 646 columns - 1 fixed gives 1668, 644 - still too large Cbc0010I
After 600 nodes, 297 on tree, 1061 best solution, best possible 682.11199
(87.16 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to
1662 rows 640 columns - 1 fixed gives 1659, 638 - still too large Cbc0010I
After 700 nodes, 348 on tree, 1061 best solution, best possible 682.11199
(91.58 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to
1677 rows 650 columns - 1 fixed gives 1674, 648 - still too large Cbc0010I
After 800 nodes, 391 on tree, 1061 best solution, best possible 682.11199
(96.49 seconds) Cbc0010I After 900 nodes, 443 on tree, 1061 best solution,
best possible 682.11199 (100.81 seconds) Cbc0010I After 1000 nodes, 487 on
tree, 1061 best solution, best possible 682.11199 (104.82 seconds)
Cbc0004I Integer solution of 956 found after 205226 iterations and 1051
nodes (107.19 seconds) Cbc0010I After 1100 nodes, 449 on tree, 956 best
solution, best possible 682.11199 (111.57 seconds) Cbc0038I Full problem
2890 rows 1451 columns, reduced to 1618 rows 602 columns - 1 fixed gives
1615, 600 - still too large Cbc0038I Full problem 2890 rows 1451 columns,
reduced to 1613 rows 599 columns - too large Cbc0010I After 1200 nodes,
493 on tree, 956 best solution, best possible 682.11199 (117.94 seconds)
Cbc0010I After 1300 nodes, 537 on tree, 956 best solution, best possible
682.11199 (124.43 seconds) Cbc0010I After 1400 nodes, 584 on tree, 956
best solution, best possible 682.11199 (131.25 seconds) Cbc0010I After
1500 nodes, 633 on tree, 956 best solution, best possible 682.11199
(137.03 seconds) Cbc0010I After 1600 nodes, 679 on tree, 956 best
solution, best possible 682.11199 (142.13 seconds) Cbc0010I After 1700
```

```
nodes, 728 on tree, 956 best solution, best possible 682.11199 (148.08
seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1614
rows 599 columns - 1 fixed gives 1611, 597 - still too large Cbc0038I Full
problem 2890 rows 1451 columns, reduced to 1607 rows 595 columns - too
large Cbc0010I After 1800 nodes, 773 on tree, 956 best solution, best
possible 682.11199 (152.37 seconds) Cbc0010I After 1900 nodes, 816 on
tree, 956 best solution, best possible 682.11199 (156.69 seconds) Cbc0010I
After 2000 nodes, 860 on tree, 956 best solution, best possible 682.11199
(161.81 seconds) Cbc0010I After 2100 nodes, 906 on tree, 956 best
solution, best possible 682.11199 (166.74 seconds) Cbc0010I After 2200
nodes, 950 on tree, 956 best solution, best possible 682.11199 (172.31
seconds) Cbc0004I Integer solution of 897 found after 408219 iterations
and 2218 nodes (172.75 seconds) Cbc0010I After 2300 nodes, 587 on tree,
897 best solution, best possible 682.11199 (192.91 seconds) Cbc0038I Full
problem 2890 rows 1451 columns, reduced to 1821 rows 746 columns - 1 fixed
gives 1818, 744 - still too large Cbc0038I Full problem 2890 rows 1451
columns, reduced to 1810 rows 740 columns - too large Cbc0010I After 2400
nodes, 638 on tree, 897 best solution, best possible 682.11199 (206.71
seconds) Cbc0010I After 2500 nodes, 678 on tree, 897 best solution, best
possible 682.11199 (216.18 seconds) Cbc0010I After 2600 nodes, 718 on
tree, 897 best solution, best possible 682.11199 (222.99 seconds) Cbc0010I
After 2700 nodes, 764 on tree, 897 best solution, best possible 682.11199
(230.27 seconds) Cbc0010I After 2800 nodes, 811 on tree, 897 best
solution, best possible 682.11199 (238.46 seconds) Cbc0010I After 2900
nodes, 834 on tree, 897 best solution, best possible 682.11199 (244.36
seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1857
rows 770 columns - 4 fixed gives 1845, 762 - still too large Cbc0038I Full
problem 2890 rows 1451 columns, reduced to 1837 rows 758 columns - too
large Cbc0010I After 3000 nodes, 879 on tree, 897 best solution, best
possible 682.11199 (252.88 seconds) Cbc0010I After 3100 nodes, 914 on
tree, 897 best solution, best possible 682.11199 (260.71 seconds) Cbc0010I
After 3200 nodes, 960 on tree, 897 best solution, best possible 682.11199
(270.26 seconds) Cbc0010I After 3300 nodes, 1003 on tree, 897 best
solution, best possible 682.11199 (280.57 seconds) Cbc0010I After 3400
nodes, 1052 on tree, 897 best solution, best possible 682.11199 (289.26
seconds) Cbc0010I After 3500 nodes, 1096 on tree, 897 best solution, best
possible 682.11199 (296.03 seconds) Cbc0038I Full problem 2890 rows 1451
columns, reduced to 1725 rows 682 columns - 2 fixed gives 1719, 678 -
still too large Cbc0038I Full problem 2890 rows 1451 columns, reduced to
1711 rows 674 columns - too large Cbc0010I After 3600 nodes, 1120 on tree,
897 best solution, best possible 682.11199 (299.83 seconds) Cbc0010I After
3700 nodes, 1151 on tree, 897 best solution, best possible 682.11199
(304.81 seconds) Cbc0010I After 3800 nodes, 1189 on tree, 897 best
solution, best possible 682.11199 (312.51 seconds) Cbc0010I After 3900
nodes, 1231 on tree, 897 best solution, best possible 682.11199 (319.32
seconds) Cbc0010I After 4000 nodes, 1273 on tree, 897 best solution, best
possible 682.11199 (325.60 seconds) Cbc0010I After 4100 nodes, 1309 on
tree, 897 best solution, best possible 682.11199 (330.28 seconds) Cbc0038I
Full problem 2890 rows 1451 columns, reduced to 1656 rows 636 columns - 1
fixed gives 1653, 634 - still too large Cbc0038I Full problem 2890 rows
1451 columns, reduced to 1645 rows 630 columns - too large Cbc0010I After
4200 nodes, 1328 on tree, 897 best solution, best possible 682.11199
(333.68 seconds) Cbc0010I After 4300 nodes, 1368 on tree, 897 best
solution, best possible 682.11199 (338.49 seconds) Cbc0010I After 4400
nodes, 1401 on tree, 897 best solution, best possible 682.11199 (343.07
seconds) Cbc0010I After 4500 nodes, 1437 on tree, 897 best solution, best
possible 682.11199 (347.56 seconds) Cbc0010I After 4600 nodes, 1472 on
tree, 897 best solution, best possible 682.11199 (353.37 seconds) Cbc0010I
After 4700 nodes, 1514 on tree, 897 best solution, best possible 682.11199
(362.49 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to
1753 rows 701 columns - 1 fixed gives 1750, 699 - still too large Cbc0038I
Full problem 2890 rows 1451 columns, reduced to 1744 rows 696 columns -
too large Cbc0010I After 4800 nodes, 1553 on tree, 897 best solution, best
possible 682.11199 (368.94 seconds) Cbc0010I After 4900 nodes, 1603 on
tree, 897 best solution, best possible 682.11199 (376.35 seconds) Cbc0010I
After 5000 nodes, 1648 on tree, 897 best solution, best possible 682.11199
(384.04 seconds) Cbc0010I After 5100 nodes, 1628 on tree, 897 best
solution, best possible 682.11199 (385.64 seconds) Cbc0010I After 5200
nodes, 1613 on tree, 897 best solution, best possible 682.11199 (387.53
```

nodes, 1515 on tree, 889 best solution, best possible 682.11199 (387.55 seconds) Cbc0004I Integer solution of 889 found after 1133511 iterations and 5214 nodes (387.76 seconds) Cbc0010I After 5300 nodes, 1549 on tree, 889 best solution, best possible 682.11199 (395.63 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1803 rows 734 columns - 3 fixed gives 1794, 728 - still too large Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1786 rows 724 columns - too large Cbc0010I After 5400 nodes, 1597 on tree, 889 best solution, best possible 682.11199 (402.52 seconds) Cbc0010I After 5500 nodes, 1642 on tree, 889 best solution, best possible 682.11199 (408.40 seconds) Cbc0010I After 5600 nodes, 1688 on tree, 889 best solution, best possible 682.11199 (415.23 seconds) Cbc0010I After 5700 nodes, 1734 on tree, 889 best solution, best possible 682.11199 (421.17 seconds) Cbc0010I After 5800 nodes, 1765 on tree, 889 best solution, best possible 682.11199 (426.11 seconds) Cbc0010I After 5900 nodes, 1806 on tree, 889 best solution, best possible 682.11199 (430.59 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1829 rows 752 columns - 3 fixed gives 1820, 746 - still too large Cbc0010I After 6000 nodes, 1849 on tree, 889 best solution, best possible 682.11199 (438.81 seconds) Cbc0010I After 6100 nodes, 1895 on tree, 889 best solution, best possible 682.11199 (447.88 seconds) Cbc0010I After 6200 nodes, 1935 on tree, 889 best solution, best possible 682.11199 (453.79 seconds) Cbc0010I After 6300 nodes, 1980 on tree, 889 best solution, best possible 682.11199 (460.38 seconds) Cbc0010I After 6400 nodes, 2026 on tree, 889 best solution, best possible 682.11199 (467.50 seconds) Cbc0010I After 6500 nodes, 2077 on tree, 889 best solution, best possible 682.11199 (474.42 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1774 rows 715 columns - 4 fixed gives 1762, 707 - still too large Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1756 rows 704 columns - too large Cbc0010I After 6600 nodes, 2127 on tree, 889 best solution, best possible 682.11199 (480.82 seconds) Cbc0010I After 6700 nodes, 2170 on tree, 889 best solution, best possible 682.11199 (488.06 seconds) Cbc0010I After 6800 nodes, 2214 on tree, 889 best solution, best possible 682.11199 (495.06 seconds) Cbc0010I After 6900 nodes, 2252 on tree, 889 best solution, best possible 682.11199 (500.87 seconds) Cbc0010I After 7000 nodes, 2293 on tree, 889 best solution, best possible 682.11199 (506.50 seconds) Cbc0010I After 7100 nodes, 2334 on tree, 889 best solution, best possible 682.11199 (512.29 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1783 rows 721 columns - 1 fixed gives 1780, 719 - still too large Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1774 rows 716 columns - too large Cbc0010I After 7200 nodes, 2378 on tree, 889 best solution, best possible 682.11199 (518.48 seconds) Cbc0010I After 7300 nodes, 2416 on tree, 889 best solution, best possible 682.11199 (523.77 seconds) Cbc0010I After 7400 nodes, 2460 on tree, 889 best solution, best possible 682.11199 (530.39 seconds) Cbc0010I After 7500 nodes, 2501 on tree, 889 best solution, best possible 682.11199 (537.20 seconds) Cbc0010I After 7600 nodes, 2540 on tree, 889 best solution, best possible 682.11199 (542.48 seconds) Cbc0010I After 7700 nodes, 2581 on tree, 889 best solution, best possible 682.11199 (547.17 seconds) Cbc0010I After 7800 nodes, 2615 on tree, 889 best solution, best possible 682.11199 (551.03 seconds) Cbc0010I After 7900 nodes, 2642 on tree, 889 best solution, best possible 682.11199 (555.95 seconds) Cbc0010I After 8000 nodes, 2677 on tree, 889 best solution, best possible 682.11199 (560.56 seconds) Cbc0010I After 8100 nodes, 2705 on tree, 889 best solution, best possible 682.11199 (565.32 seconds) Cbc0010I After 8200 nodes, 2747 on tree, 889 best solution, best possible 682.11199 (573.65 seconds) Cbc0010I After 8300 nodes, 2780 on tree, 889 best solution, best possible 682.11199 (578.17 seconds) Cbc0010I After 8400 nodes, 2806 on tree, 889 best solution, best possible 682.11199 (582.03 seconds) Cbc0010I After 8500 nodes, 2832 on tree, 889 best solution, best possible 682.11199 (585.60 seconds) Cbc0010I After 8600 nodes, 2865 on tree, 889 best solution, best possible 682.11199 (590.66 seconds) Cbc0010I After 8700 nodes, 2905 on tree, 889 best solution, best possible 682.11199 (597.21 seconds) Cbc0010I After 8800 nodes, 2953 on tree, 889 best solution, best possible 682.11199 (605.35 seconds) Cbc0010I After 8900 nodes, 2996 on tree, 889 best solution, best possible 682.11199 (613.53 seconds) Cbc0010I After 9000 nodes, 3029 on tree, 889 best solution, best possible 682.11199 (621.04 seconds) Cbc0010I After 9100 nodes, 3075 on tree, 889 best solution, best possible 682.11199 (627.92 seconds) Cbc0010I After 9200 nodes, 3066 on tree, 889 best solution, best possible 682.11199 (631.29 seconds) Cbc0010I

```
After 9300 nodes, 3077 on tree, 889 best solution, best possible 682.11199
(634.53 seconds) Cbc0010I After 9400 nodes, 3066 on tree, 889 best
solution, best possible 682.11199 (637.27 seconds) Cbc0010I After 9500
nodes, 3060 on tree, 889 best solution, best possible 682.11199 (640.13
seconds) Cbc0010I After 9600 nodes, 3074 on tree, 889 best solution, best
possible 682.11199 (643.69 seconds) Cbc0010I After 9700 nodes, 3061 on
tree, 889 best solution, best possible 682.11199 (646.84 seconds) Cbc0010I
After 9800 nodes, 3076 on tree, 889 best solution, best possible 682.11199
(649.77 seconds) Cbc0010I After 9900 nodes, 3071 on tree, 889 best
solution, best possible 682.11199 (652.28 seconds) Cbc0010I After 10000
nodes, 3072 on tree, 889 best solution, best possible 682.11199 (654.44
seconds) Cbc0010I After 10100 nodes, 3074 on tree, 889 best solution, best
possible 682.11199 (657.05 seconds) Cbc0010I After 10200 nodes, 3071 on
tree, 889 best solution, best possible 682.11199 (660.70 seconds) Cbc0010I
After 10300 nodes, 3071 on tree, 889 best solution, best possible
682.11199 (663.65 seconds) Cbc0010I After 10400 nodes, 3076 on tree, 889
best solution, best possible 682.11199 (666.72 seconds) Cbc0010I After
10500 nodes, 3068 on tree, 889 best solution, best possible 682.11199
(669.80 seconds) Cbc0010I After 10600 nodes, 3066 on tree, 889 best
solution, best possible 682.11199 (672.84 seconds) Cbc0010I After 10700
nodes, 3066 on tree, 889 best solution, best possible 682.11199 (676.16
seconds) Cbc0010I After 10800 nodes, 3076 on tree, 889 best solution, best
possible 682.11199 (680.44 seconds) Cbc0010I After 10900 nodes, 3067 on
tree, 889 best solution, best possible 682.11199 (683.40 seconds) Cbc0010I
After 11000 nodes, 3073 on tree, 889 best solution, best possible
682.11199 (686.38 seconds) Cbc0010I After 11100 nodes, 3123 on tree, 889
best solution, best possible 686.18691 (751.86 seconds) Cbc0010I After
11200 nodes, 3167 on tree, 889 best solution, best possible 700.41015
(791.68 seconds) Cbc0010I After 11300 nodes, 3217 on tree, 889 best
solution, best possible 711.83532 (818.62 seconds) Cbc0010I After 11400
nodes, 3261 on tree, 889 best solution, best possible 715.25415 (847.23
seconds) Cbc0010I After 11500 nodes, 3310 on tree, 889 best solution, best
possible 718.8264 (866.97 seconds) Cbc0010I After 11600 nodes, 3355 on
tree, 889 best solution, best possible 722.46862 (882.54 seconds) Cbc0010I
After 11700 nodes, 3405 on tree, 889 best solution, best possible 723.3876
(898.20 seconds) Cbc0010I After 11800 nodes, 3455 on tree, 889 best
solution, best possible 724.10366 (912.29 seconds) Cbc0010I After 11900
nodes, 3505 on tree, 889 best solution, best possible 725.79412 (928.09
seconds) Cbc0010I After 12000 nodes, 3555 on tree, 889 best solution, best
possible 726.70413 (941.71 seconds) Cbc0010I After 12100 nodes, 3602 on
tree, 889 best solution, best possible 728.00778 (960.26 seconds) Cbc0010I
After 12200 nodes, 3652 on tree, 889 best solution, best possible
728.54962 (978.17 seconds) Cbc0010I After 12300 nodes, 3700 on tree, 889
best solution, best possible 728.72002 (988.83 seconds) Cbc0010I After
12400 nodes, 3750 on tree, 889 best solution, best possible 728.99992
(1006.50 seconds) Cbc0010I After 12500 nodes, 3800 on tree, 889 best
solution, best possible 729.0175 (1025.90 seconds) Cbc0010I After 12600
nodes, 3849 on tree, 889 best solution, best possible 729.05658 (1041.18
seconds) Cbc0010I After 12700 nodes, 3898 on tree, 889 best solution, best
possible 729.11291 (1054.89 seconds) Cbc0010I After 12800 nodes, 3945 on
tree, 889 best solution, best possible 729.23024 (1069.36 seconds)
Cbc0010I After 12900 nodes, 3994 on tree, 889 best solution, best possible
729.26201 (1083.11 seconds) Cbc0010I After 13000 nodes, 4042 on tree, 889
best solution, best possible 729.45193 (1096.44 seconds) Cbc0010I After
13100 nodes, 4069 on tree, 889 best solution, best possible 729.45193
(1100.43 seconds) Cbc0010I After 13200 nodes, 4060 on tree, 889 best
solution, best possible 729.45193 (1103.07 seconds) Cbc0010I After 13300
nodes, 4060 on tree, 889 best solution, best possible 729.45193 (1105.58
seconds) Cbc0010I After 13400 nodes, 4058 on tree, 889 best solution, best
possible 729.45193 (1108.24 seconds) Cbc0010I After 13500 nodes, 4061 on
tree, 889 best solution, best possible 729.45193 (1111.28 seconds)
Cbc0010I After 13600 nodes, 4055 on tree, 889 best solution, best possible
729.45193 (1114.01 seconds) Cbc0010I After 13700 nodes, 4060 on tree, 889
best solution, best possible 729.45193 (1117.31 seconds) Cbc0010I After
13800 nodes, 4053 on tree, 889 best solution, best possible 729.45193
(1120.99 seconds) Cbc0010I After 13900 nodes, 4050 on tree, 889 best
solution, best possible 729.45193 (1124.83 seconds) Cbc0010I After 14000
nodes, 4056 on tree, 889 best solution, best possible 729.45193 (1127.83
seconds) Cbc0010I After 14100 nodes, 4105 on tree, 889 best solution, best
```

```
seconds) Cbc0010I After 14100 nodes, 4105 on tree, 889 best solution, best
possible 729.67153 (1140.12 seconds) Cbc0010I After 14200 nodes, 4153 on
tree, 889 best solution, best possible 729.75885 (1153.37 seconds)
Cbc0010I After 14300 nodes, 4201 on tree, 889 best solution, best possible
729.90514 (1165.94 seconds) Cbc0038I Full problem 2890 rows 1451 columns,
reduced to 1952 rows 833 columns - 1 fixed gives 1949, 831 - still too
large Cbc0010I After 14400 nodes, 4251 on tree, 889 best solution, best
possible 730.02826 (1177.60 seconds) Cbc0010I After 14500 nodes, 4299 on
tree, 889 best solution, best possible 730.32972 (1191.06 seconds)
Cbc0010I After 14600 nodes, 4347 on tree, 889 best solution, best possible
730.52212 (1202.16 seconds) Cbc0010I After 14700 nodes, 4395 on tree, 889
best solution, best possible 730.52215 (1211.07 seconds) Cbc0010I After
14800 nodes, 4444 on tree, 889 best solution, best possible 730.75517
(1224.02 seconds) Cbc0010I After 14900 nodes, 4492 on tree, 889 best
solution, best possible 730.76288 (1234.17 seconds) Cbc0010I After 15000
nodes, 4540 on tree, 889 best solution, best possible 730.76452 (1243.83
seconds) Cbc0010I After 15100 nodes, 4579 on tree, 889 best solution, best
possible 730.76491 (1253.21 seconds) Cbc0010I After 15200 nodes, 4627 on
tree, 889 best solution, best possible 730.88663 (1264.42 seconds)
Cbc0010I After 15300 nodes, 4675 on tree, 889 best solution, best possible
731.12728 (1278.40 seconds) Cbc0010I After 15400 nodes, 4725 on tree, 889
best solution, best possible 731.70567 (1293.29 seconds) Cbc0010I After
15500 nodes, 4774 on tree, 889 best solution, best possible 731.88981
(1304.89 seconds) Cbc0010I After 15600 nodes, 4823 on tree, 889 best
solution, best possible 732.09451 (1317.70 seconds) Cbc0010I After 15700
nodes, 4872 on tree, 889 best solution, best possible 732.26423 (1328.91
seconds) Cbc0010I After 15800 nodes, 4922 on tree, 889 best solution, best
possible 732.29782 (1337.26 seconds) Cbc0010I After 15900 nodes, 4972 on
tree, 889 best solution, best possible 732.43945 (1349.34 seconds)
Cbc0010I After 16000 nodes, 5020 on tree, 889 best solution, best possible
732.66332 (1361.22 seconds) Cbc0010I After 16100 nodes, 5061 on tree, 889
best solution, best possible 732.70552 (1369.41 seconds) Cbc0010I After
16200 nodes, 5105 on tree, 889 best solution, best possible 732.77095
(1379.22 seconds) Cbc0010I After 16300 nodes, 5154 on tree, 889 best
solution, best possible 732.84891 (1392.29 seconds) Cbc0010I After 16400
nodes, 5202 on tree, 889 best solution, best possible 732.99927 (1405.42
seconds) Cbc0010I After 16500 nodes, 5252 on tree, 889 best solution, best
possible 733 (1421.74 seconds) Cbc0010I After 16600 nodes, 5300 on tree,
889 best solution, best possible 733 (1437.09 seconds) Cbc0010I After
16700 nodes, 5348 on tree, 889 best solution, best possible 733.07999
(1453.39 seconds) Cbc0010I After 16800 nodes, 5392 on tree, 889 best
solution, best possible 733.10336 (1465.95 seconds) Cbc0010I After 16900
nodes, 5441 on tree, 889 best solution, best possible 733.10336 (1477.46
seconds) Cbc0010I After 17000 nodes, 5489 on tree, 889 best solution, best
possible 733.10338 (1488.87 seconds) Cbc0010I After 17100 nodes, 5505 on
tree, 889 best solution, best possible 733.10338 (1492.93 seconds)
Cbc0010I After 17200 nodes, 5497 on tree, 889 best solution, best possible
733.10338 (1497.12 seconds) Cbc0010I After 17300 nodes, 5523 on tree, 889
best solution, best possible 733.10338 (1500.93 seconds) Cbc0010I After
17400 nodes, 5507 on tree, 889 best solution, best possible 733.10338
(1503.52 seconds) Cbc0010I After 17500 nodes, 5504 on tree, 889 best
solution, best possible 733.10338 (1506.32 seconds) Cbc0016I Integer
solution of 882 found by strong branching after 4993588 iterations and
17553 nodes (1507.69 seconds) Cbc0010I After 17600 nodes, 5277 on tree,
882 best solution, best possible 733.10338 (1512.99 seconds) Cbc0010I
After 17700 nodes, 5323 on tree, 882 best solution, best possible
733.10338 (1523.10 seconds) Cbc0010I After 17800 nodes, 5370 on tree, 882
best solution, best possible 733.10338 (1533.36 seconds) Cbc0010I After
17900 nodes, 5419 on tree, 882 best solution, best possible 733.10338
(1544.10 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to
1899 rows 798 columns - 2 fixed gives 1893, 794 - still too large Cbc0010I
After 18000 nodes, 5467 on tree, 882 best solution, best possible
733.10338 (1554.88 seconds) Cbc0010I After 18100 nodes, 5516 on tree, 882
best solution, best possible 733.3401 (1568.29 seconds) Cbc0010I After
18200 nodes, 5566 on tree, 882 best solution, best possible 733.46123
(1581.49 seconds) Cbc0010I After 18300 nodes, 5614 on tree, 882 best
solution, best possible 733.57827 (1593.84 seconds) Cbc0010I After 18400
nodes, 5661 on tree, 882 best solution, best possible 733.80484 (1607.33
seconds) Cbc0010I After 18500 nodes, 5710 on tree, 882 best solution, best
```

```
possible 734.08726 (1620.18 seconds) Cbc0010I After 18600 nodes, 5757 on
tree, 882 best solution, best possible 734.20593 (1632.05 seconds)
Cbc0010I After 18700 nodes, 5806 on tree, 882 best solution, best possible
734.44881 (1643.71 seconds) Cbc0010I After 18800 nodes, 5855 on tree, 882
best solution, best possible 734.62801 (1656.89 seconds) Cbc0010I After
18900 nodes, 5904 on tree, 882 best solution, best possible 734.71231
(1671.00 seconds) Cbc0010I After 19000 nodes, 5950 on tree, 882 best
solution, best possible 734.82349 (1683.76 seconds) Cbc0010I After 19100
nodes, 5999 on tree, 882 best solution, best possible 735.07555 (1695.65
seconds) Cbc0010I After 19200 nodes, 6045 on tree, 882 best solution, best
possible 735.29913 (1708.17 seconds) Cbc0010I After 19300 nodes, 6092 on
tree, 882 best solution, best possible 735.60607 (1720.55 seconds)
Cbc0010I After 19400 nodes, 6141 on tree, 882 best solution, best possible
735.86852 (1733.62 seconds) Cbc0010I After 19500 nodes, 6190 on tree, 882
best solution, best possible 736.07229 (1749.52 seconds) Cbc0010I After
19600 nodes, 6238 on tree, 882 best solution, best possible 736.2792
(1762.00 seconds) Cbc0010I After 19700 nodes, 6288 on tree, 882 best
solution, best possible 736.41781 (1774.57 seconds) Cbc0010I After 19800
nodes, 6338 on tree, 882 best solution, best possible 736.69957 (1785.89
seconds) Cbc0010I After 19900 nodes, 6386 on tree, 882 best solution, best
possible 736.90498 (1798.08 seconds) Cbc0010I After 20000 nodes, 6435 on
tree, 882 best solution, best possible 737.12255 (1811.84 seconds)
Cbc0010I After 20100 nodes, 6483 on tree, 882 best solution, best possible
737.46756 (1824.72 seconds) Cbc0010I After 20200 nodes, 6531 on tree, 882
best solution, best possible 737.75776 (1836.25 seconds) Cbc0010I After
20300 nodes, 6579 on tree, 882 best solution, best possible 737.93748
(1846.52 seconds) Cbc0010I After 20400 nodes, 6627 on tree, 882 best
solution, best possible 738.08643 (1860.42 seconds) Cbc0010I After 20500
nodes, 6677 on tree, 882 best solution, best possible 738.21701 (1873.98
seconds) Cbc0010I After 20600 nodes, 6723 on tree, 882 best solution, best
possible 738.53513 (1888.31 seconds) Cbc0010I After 20700 nodes, 6769 on
tree, 882 best solution, best possible 738.73596 (1900.05 seconds)
Cbc0010I After 20800 nodes, 6818 on tree, 882 best solution, best possible
738.96624 (1912.10 seconds) Cbc0010I After 20900 nodes, 6867 on tree, 882
best solution, best possible 739 (1924.70 seconds) Cbc0010I After 21000
nodes, 6913 on tree, 882 best solution, best possible 739.115 (1939.04
seconds) Cbc0010I After 21100 nodes, 6917 on tree, 882 best solution, best
possible 739.115 (1942.67 seconds) Cbc0010I After 21200 nodes, 6927 on
tree, 882 best solution, best possible 739.115 (1947.10 seconds) Cbc0010I
After 21300 nodes, 6924 on tree, 882 best solution, best possible 739.115
(1949.91 seconds) Cbc0010I After 21400 nodes, 6920 on tree, 882 best
solution, best possible 739.115 (1952.50 seconds) Cbc0010I After 21500
nodes, 6928 on tree, 882 best solution, best possible 739.115 (1954.79
seconds) Cbc0010I After 21600 nodes, 6924 on tree, 882 best solution, best
possible 739.115 (1956.62 seconds) Cbc0010I After 21700 nodes, 6921 on
tree, 882 best solution, best possible 739.115 (1960.12 seconds) Cbc0010I
After 21800 nodes, 6923 on tree, 882 best solution, best possible 739.115
(1964.12 seconds) Cbc0010I After 21900 nodes, 6922 on tree, 882 best
solution, best possible 739.115 (1967.49 seconds) Cbc0010I After 22000
nodes, 6922 on tree, 882 best solution, best possible 739.115 (1970.57
seconds) Cbc0010I After 22100 nodes, 6972 on tree, 882 best solution, best
possible 739.24296 (1986.87 seconds) Cbc0010I After 22200 nodes, 7017 on
tree, 882 best solution, best possible 739.37994 (2003.74 seconds)
Cbc0010I After 22300 nodes, 7064 on tree, 882 best solution, best possible
739.60326 (2017.23 seconds) Cbc0010I After 22400 nodes, 7111 on tree, 882
best solution, best possible 740 (2035.32 seconds) Cbc0010I After 22500
nodes, 7161 on tree, 882 best solution, best possible 740.24006 (2050.10
seconds) Cbc0010I After 22600 nodes, 7210 on tree, 882 best solution, best
possible 740.46446 (2061.60 seconds) Cbc0010I After 22700 nodes, 7258 on
tree, 882 best solution, best possible 740.76773 (2075.83 seconds)
Cbc0010I After 22800 nodes, 7307 on tree, 882 best solution, best possible
740.99991 (2088.74 seconds) Cbc0010I After 22900 nodes, 7356 on tree, 882
best solution, best possible 741.19873 (2104.03 seconds) Cbc0010I After
23000 nodes, 7404 on tree, 882 best solution, best possible 741.49656
(2117.32 seconds) Cbc0010I After 23100 nodes, 7452 on tree, 882 best
solution, best possible 741.73392 (2133.11 seconds) Cbc0010I After 23200
nodes, 7501 on tree, 882 best solution, best possible 741.99774 (2147.38
seconds) Cbc0010I After 23300 nodes, 7551 on tree, 882 best solution, best
```

```
possible 742.06794 (2161.51 seconds) Cbc0010I After 23400 nodes, 7601 on
tree, 882 best solution, best possible 742.32271 (2175.24 seconds)
Cbc0010I After 23500 nodes, 7650 on tree, 882 best solution, best possible
742.48402 (2189.04 seconds) Cbc0010I After 23600 nodes, 7699 on tree, 882
best solution, best possible 742.77206 (2202.06 seconds) Cbc0010I After
23700 nodes, 7746 on tree, 882 best solution, best possible 742.88849
(2215.79 seconds) Cbc0010I After 23800 nodes, 7793 on tree, 882 best
solution, best possible 743.08754 (2229.42 seconds) Cbc0010I After 23900
nodes, 7842 on tree, 882 best solution, best possible 743.32798 (2242.52
seconds) Cbc0010I After 24000 nodes, 7889 on tree, 882 best solution, best
possible 743.58079 (2254.53 seconds) Cbc0010I After 24100 nodes, 7938 on
tree, 882 best solution, best possible 743.84176 (2267.73 seconds)
Cbc0010I After 24200 nodes, 7986 on tree, 882 best solution, best possible
744.02519 (2281.98 seconds) Cbc0010I After 24300 nodes, 8034 on tree, 882
best solution, best possible 744.22381 (2296.40 seconds) Cbc0010I After
24400 nodes, 8081 on tree, 882 best solution, best possible 744.56534
(2309.17 seconds) Cbc0010I After 24500 nodes, 8129 on tree, 882 best
solution, best possible 744.81551 (2323.49 seconds) Cbc0010I After 24600
nodes, 8173 on tree, 882 best solution, best possible 744.93327 (2334.52
seconds) Cbc0010I After 24700 nodes, 8220 on tree, 882 best solution, best
possible 744.99929 (2346.90 seconds) Cbc0010I After 24800 nodes, 8268 on
tree, 882 best solution, best possible 744.99974 (2359.83 seconds)
Cbc0010I After 24900 nodes, 8317 on tree, 882 best solution, best possible
744.99979 (2375.28 seconds) Cbc0010I After 25000 nodes, 8367 on tree, 882
best solution, best possible 744.9998 (2385.34 seconds) Cbc0010I After
25100 nodes, 8379 on tree, 882 best solution, best possible 744.9998
(2390.18 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to
1799 rows 732 columns - 2 fixed gives 1793, 728 - still too large Cbc0010I
After 25200 nodes, 8381 on tree, 882 best solution, best possible 744.9998
(2394.11 seconds) Cbc0010I After 25300 nodes, 8375 on tree, 882 best
solution, best possible 744.9998 (2398.91 seconds) Cbc0010I After 25400
nodes, 8370 on tree, 882 best solution, best possible 744.9998 (2404.02
seconds) Cbc0010I After 25500 nodes, 8375 on tree, 882 best solution, best
possible 744.9998 (2408.97 seconds) Cbc0010I After 25600 nodes, 8373 on
tree, 882 best solution, best possible 744.9998 (2413.47 seconds) Cbc0010I
After 25700 nodes, 8384 on tree, 882 best solution, best possible 744.9998
(2417.12 seconds) Cbc0010I After 25800 nodes, 8384 on tree, 882 best
solution, best possible 744.9998 (2420.25 seconds) Cbc0010I After 25900
nodes, 8379 on tree, 882 best solution, best possible 744.9998 (2423.10
seconds) Cbc0010I After 26000 nodes, 8378 on tree, 882 best solution, best
possible 744.9998 (2426.36 seconds) Cbc0010I After 26100 nodes, 8428 on
tree, 882 best solution, best possible 744.99983 (2439.28 seconds)
Cbc0010I After 26200 nodes, 8476 on tree, 882 best solution, best possible
745 (2452.64 seconds) Cbc0010I After 26300 nodes, 8524 on tree, 882 best
solution, best possible 745.00488 (2465.77 seconds) Cbc0010I After 26400
nodes, 8571 on tree, 882 best solution, best possible 745.03137 (2477.56
seconds) Cbc0010I After 26500 nodes, 8619 on tree, 882 best solution, best
possible 745.21208 (2489.63 seconds) Cbc0010I After 26600 nodes, 8664 on
tree, 882 best solution, best possible 745.33924 (2502.24 seconds)
Cbc0010I After 26700 nodes, 8706 on tree, 882 best solution, best possible
745.41945 (2512.23 seconds) Cbc0010I After 26800 nodes, 8753 on tree, 882
best solution, best possible 745.64595 (2525.28 seconds) Cbc0010I After
26900 nodes, 8801 on tree, 882 best solution, best possible 745.82833
(2537.43 seconds) Cbc0010I After 27000 nodes, 8849 on tree, 882 best
solution, best possible 746.0656 (2551.34 seconds) Cbc0010I After 27100
nodes, 8897 on tree, 882 best solution, best possible 746.23246 (2564.95
seconds) Cbc0010I After 27200 nodes, 8944 on tree, 882 best solution, best
possible 746.37669 (2577.08 seconds) Cbc0010I After 27300 nodes, 8990 on
tree, 882 best solution, best possible 746.46401 (2598.91 seconds)
Cbc0010I After 27400 nodes, 9034 on tree, 882 best solution, best possible
746.48485 (2609.84 seconds) Cbc0010I After 27500 nodes, 9083 on tree, 882
best solution, best possible 746.59214 (2621.37 seconds) Cbc0010I After
27600 nodes, 9129 on tree, 882 best solution, best possible 746.77434
(2634.25 seconds) Cbc0010I After 27700 nodes, 9174 on tree, 882 best
solution, best possible 746.97052 (2646.12 seconds) Cbc0010I After 27800
nodes, 9221 on tree, 882 best solution, best possible 747 (2660.41
seconds) Cbc0010I After 27900 nodes, 9268 on tree, 882 best solution, best
possible 747.12041 (2673.07 seconds) Cbc0010I After 28000 nodes, 9315 on
tree, 882 best solution, best possible 747.24282 (2685.60 seconds)
```

[illegible] Cbc0010I After 28100 nodes, 9363 on tree, 882 best solution, best possible 747.3398 (2697.46 seconds) Cbc0010I After 28200 nodes, 9411 on tree, 882 best solution, best possible 747.43881 (2709.62 seconds) Cbc0010I After 28300 nodes, 9455 on tree, 882 best solution, best possible 747.63329 (2722.04 seconds) Cbc0010I After 28400 nodes, 9502 on tree, 882 best solution, best possible 747.8581 (2735.72 seconds) Cbc0010I After 28500 nodes, 9549 on tree, 882 best solution, best possible 747.99928 (2748.03 seconds) Cbc0010I After 28600 nodes, 9596 on tree, 882 best solution, best possible 748.04545 (2759.53 seconds) Cbc0010I After 28700 nodes, 9642 on tree, 882 best solution, best possible 748.2553 (2773.36 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 2041 rows 883 columns - 1 fixed gives 2038, 881 - still too large Cbc0010I After 28800 nodes, 9690 on tree, 882 best solution, best possible 748.41111 (2786.39 seconds) Cbc0010I After 28900 nodes, 9737 on tree, 882 best solution, best possible 748.57325 (2800.52 seconds) Cbc0010I After 29000 nodes, 9784 on tree, 882 best solution, best possible 748.67239 (2811.61 seconds) Cbc0010I After 29100 nodes, 9790 on tree, 882 best solution, best possible 748.67239 (2816.67 seconds) Cbc0010I After 29200 nodes, 9789 on tree, 882 best solution, best possible 748.67239 (2820.03 seconds) Cbc0010I After 29300 nodes, 9800 on tree, 882 best solution, best possible 748.67239 (2824.65 seconds) Cbc0010I After 29400 nodes, 9794 on tree, 882 best solution, best possible 748.67239 (2828.17 seconds) Cbc0010I After 29500 nodes, 9792 on tree, 882 best solution, best possible 748.67239 (2832.67 seconds) Cbc0010I After 29600 nodes, 9788 on tree, 882 best solution, best possible 748.67239 (2835.45 seconds) Cbc0010I After 29700 nodes, 9785 on tree, 882 best solution, best possible 748.67239 (2839.65 seconds) Cbc0010I After 29800 nodes, 9773 on tree, 882 best solution, best possible 748.67239 (2842.17 seconds) Cbc0010I After 29900 nodes, 9761 on tree, 882 best solution, best possible 748.67239 (2844.48 seconds) Cbc0010I After 30000 nodes, 9747 on tree, 882 best solution, best possible 748.67239 (2846.39 seconds) Cbc0010I After 30100 nodes, 9794 on tree, 882 best solution, best possible 748.77244 (2858.89 seconds) Cbc0010I After 30200 nodes, 9844 on tree, 882 best solution, best possible 748.89377 (2871.43 seconds) Cbc0010I After 30300 nodes, 9891 on tree, 882 best solution, best possible 748.99979 (2884.38 seconds) Cbc0010I After 30400 nodes, 9939 on tree, 882 best solution, best possible 749 (2906.00 seconds) Cbc0010I After 30500 nodes, 9981 on tree, 882 best solution, best possible 749.02192 (2926.05 seconds) Cbc0010I After 30600 nodes, 10029 on tree, 882 best solution, best possible 749.15506 (2939.29 seconds) Cbc0010I After 30700 nodes, 10080 on tree, 882 best solution, best possible 749.16161 (2951.48 seconds) Cbc0010I After 30800 nodes, 10124 on tree, 882 best solution, best possible 749.27708 (2964.89 seconds) Cbc0010I After 30900 nodes, 10167 on tree, 882 best solution, best possible 749.37811 (2978.89 seconds) Cbc0010I After 31000 nodes, 10214 on tree, 882 best solution, best possible 749.53188 (2993.11 seconds) Cbc0010I After 31100 nodes, 10260 on tree, 882 best solution, best possible 749.53188 (3004.64 seconds) Cbc0010I After 31200 nodes, 10309 on tree, 882 best solution, best possible 749.53188 (3014.48 seconds) Cbc0010I After 31300 nodes, 10357 on tree, 882 best solution, best possible 749.53188 (3024.49 seconds) Cbc0010I After 31400 nodes, 10403 on tree, 882 best solution, best possible 749.53188 (3035.61 seconds) Cbc0010I After 31500 nodes, 10449 on tree, 882 best solution, best possible 749.53188 (3045.83 seconds) Cbc0010I After 31600 nodes, 10495 on tree, 882 best solution, best possible 749.53188 (3056.29 seconds) Cbc0010I After 31700 nodes, 10544 on tree, 882 best solution, best possible 749.53188 (3067.94 seconds) Cbc0010I After 31800 nodes, 10590 on tree, 882 best solution, best possible 749.53188 (3077.88 seconds) Cbc0010I After 31900 nodes, 10638 on tree, 882 best solution, best possible 749.53188 (3088.42 seconds) Cbc0010I After 32000 nodes, 10682 on tree, 882 best solution, best possible 749.53188 (3099.48 seconds) Cbc0010I After 32100 nodes, 10675 on tree, 882 best solution, best possible 749.53188 (3101.15 seconds) Cbc0012I Integer solution of 880 found by rounding after 11235570 iterations and 32108 nodes (3101.28 seconds) Cbc0010I After 32200 nodes, 10641 on tree, 880 best solution, best possible 749.53188 (3109.67 seconds) Cbc0010I After 32300 nodes, 10684 on tree, 880 best solution, best possible 749.53188 (3120.28 seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1947 rows 820 columns - 2 fixed gives 1941, 816 - still too large Cbc0010I After 32400 nodes, 10731 on tree, 880 best

```
solution, best possible 749.53188 (3129.47 seconds) Cbc0010I After 32500
nodes, 10775 on tree, 880 best solution, best possible 749.53188 (3140.58
seconds) Cbc0010I After 32600 nodes, 10822 on tree, 880 best solution,
best possible 749.53188 (3150.13 seconds) Cbc0010I After 32700 nodes,
10866 on tree, 880 best solution, best possible 749.53188 (3160.99
seconds) Cbc0010I After 32800 nodes, 10911 on tree, 880 best solution,
best possible 749.53188 (3170.81 seconds) Cbc0010I After 32900 nodes,
10957 on tree, 880 best solution, best possible 749.53188 (3180.96
seconds) Cbc0010I After 33000 nodes, 11005 on tree, 880 best solution,
best possible 749.53188 (3191.91 seconds) Cbc0010I After 33100 nodes,
11028 on tree, 880 best solution, best possible 749.53377 (3196.50
seconds) Cbc0010I After 33200 nodes, 11027 on tree, 880 best solution,
best possible 749.53377 (3200.86 seconds) Cbc0010I After 33300 nodes,
11041 on tree, 880 best solution, best possible 749.53377 (3203.84
seconds) Cbc0010I After 33400 nodes, 11023 on tree, 880 best solution,
best possible 749.53377 (3206.66 seconds) Cbc0010I After 33500 nodes,
11024 on tree, 880 best solution, best possible 749.53377 (3210.79
seconds) Cbc0010I After 33600 nodes, 11023 on tree, 880 best solution,
best possible 749.53377 (3215.39 seconds) Cbc0010I After 33700 nodes,
11030 on tree, 880 best solution, best possible 749.53377 (3218.72
seconds) Cbc0010I After 33800 nodes, 11022 on tree, 880 best solution,
best possible 749.53377 (3221.36 seconds) Cbc0010I After 33900 nodes,
11023 on tree, 880 best solution, best possible 749.53377 (3224.80
seconds) Cbc0010I After 34000 nodes, 11029 on tree, 880 best solution,
best possible 749.53377 (3227.75 seconds) Cbc0010I After 34100 nodes,
11075 on tree, 880 best solution, best possible 749.53377 (3239.05
seconds) Cbc0010I After 34200 nodes, 11120 on tree, 880 best solution,
best possible 749.53377 (3248.81 seconds) Cbc0010I After 34300 nodes,
11163 on tree, 880 best solution, best possible 749.53377 (3260.16
seconds) Cbc0010I After 34400 nodes, 11204 on tree, 880 best solution,
best possible 749.53377 (3269.86 seconds) Cbc0010I After 34500 nodes,
11254 on tree, 880 best solution, best possible 749.53377 (3280.74
seconds) Cbc0010I After 34600 nodes, 11297 on tree, 880 best solution,
best possible 749.53377 (3291.49 seconds) Cbc0010I After 34700 nodes,
11340 on tree, 880 best solution, best possible 749.53377 (3301.86
seconds) Cbc0010I After 34800 nodes, 11381 on tree, 880 best solution,
best possible 749.53377 (3311.84 seconds) Cbc0010I After 34900 nodes,
11427 on tree, 880 best solution, best possible 749.53377 (3322.69
seconds) Cbc0010I After 35000 nodes, 11467 on tree, 880 best solution,
best possible 749.53377 (3330.93 seconds) Cbc0010I After 35100 nodes,
11515 on tree, 880 best solution, best possible 749.53377 (3342.01
seconds) Cbc0010I After 35200 nodes, 11559 on tree, 880 best solution,
best possible 749.53377 (3352.56 seconds) Cbc0010I After 35300 nodes,
11605 on tree, 880 best solution, best possible 749.53377 (3363.13
seconds) Cbc0010I After 35400 nodes, 11646 on tree, 880 best solution,
best possible 749.53377 (3372.22 seconds) Cbc0010I After 35500 nodes,
11689 on tree, 880 best solution, best possible 749.53377 (3381.46
seconds) Cbc0010I After 35600 nodes, 11732 on tree, 880 best solution,
best possible 749.53377 (3391.87 seconds) Cbc0010I After 35700 nodes,
11776 on tree, 880 best solution, best possible 749.53377 (3402.49
seconds) Cbc0010I After 35800 nodes, 11822 on tree, 880 best solution,
best possible 749.53377 (3413.52 seconds) Cbc0010I After 35900 nodes,
11867 on tree, 880 best solution, best possible 749.53377 (3422.52
seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1886
rows 789 columns - 2 fixed gives 1880, 785 - still too large Cbc0010I
After 36000 nodes, 11911 on tree, 880 best solution, best possible
749.53377 (3432.69 seconds) Cbc0010I After 36100 nodes, 11899 on tree, 880
best solution, best possible 749.53377 (3434.26 seconds) Cbc0010I After
36200 nodes, 11887 on tree, 880 best solution, best possible 749.53377
(3435.69 seconds) Cbc0010I After 36300 nodes, 11879 on tree, 880 best
solution, best possible 749.53377 (3437.34 seconds) Cbc0010I After 36400
nodes, 11868 on tree, 880 best solution, best possible 749.53377 (3438.99
seconds) Cbc0010I After 36500 nodes, 11858 on tree, 880 best solution,
best possible 749.53377 (3441.02 seconds) Cbc0010I After 36600 nodes,
11851 on tree, 880 best solution, best possible 749.53377 (3442.67
seconds) Cbc0010I After 36700 nodes, 11853 on tree, 880 best solution,
best possible 749.53377 (3444.69 seconds) Cbc0010I After 36800 nodes,
11839 on tree, 880 best solution, best possible 749.53377 (3446.81
seconds) Cbc0010I After 36900 nodes, 11827 on tree, 880 best solution
```

```
seconds) Cbc0010I After 36900 nodes, 11827 on tree, 880 best solution,
best possible 749.53377 (3449.15 seconds) Cbc0010I After 37000 nodes,
11819 on tree, 880 best solution, best possible 749.53377 (3451.03
seconds) Cbc0010I After 37100 nodes, 11838 on tree, 880 best solution,
best possible 749.53377 (3454.74 seconds) Cbc0010I After 37200 nodes,
11837 on tree, 880 best solution, best possible 749.53377 (3457.53
seconds) Cbc0010I After 37300 nodes, 11835 on tree, 880 best solution,
best possible 749.53377 (3461.33 seconds) Cbc0010I After 37400 nodes,
11833 on tree, 880 best solution, best possible 749.53377 (3465.33
seconds) Cbc0010I After 37500 nodes, 11838 on tree, 880 best solution,
best possible 749.53377 (3468.12 seconds) Cbc0010I After 37600 nodes,
11825 on tree, 880 best solution, best possible 749.53377 (3470.97
seconds) Cbc0010I After 37700 nodes, 11831 on tree, 880 best solution,
best possible 749.53377 (3474.46 seconds) Cbc0010I After 37800 nodes,
11834 on tree, 880 best solution, best possible 749.53377 (3478.49
seconds) Cbc0010I After 37900 nodes, 11825 on tree, 880 best solution,
best possible 749.53377 (3481.65 seconds) Cbc0010I After 38000 nodes,
11831 on tree, 880 best solution, best possible 749.53377 (3485.21
seconds) Cbc0010I After 38100 nodes, 11872 on tree, 880 best solution,
best possible 749.53377 (3495.96 seconds) Cbc0010I After 38200 nodes,
11918 on tree, 880 best solution, best possible 749.53377 (3506.72
seconds) Cbc0010I After 38300 nodes, 11961 on tree, 880 best solution,
best possible 749.53377 (3515.28 seconds) Cbc0010I After 38400 nodes,
12009 on tree, 880 best solution, best possible 749.53377 (3525.32
seconds) Cbc0010I After 38500 nodes, 12054 on tree, 880 best solution,
best possible 749.53377 (3536.11 seconds) Cbc0010I After 38600 nodes,
12093 on tree, 880 best solution, best possible 749.53377 (3545.49
seconds) Cbc0010I After 38700 nodes, 12137 on tree, 880 best solution,
best possible 749.53377 (3555.04 seconds) Cbc0010I After 38800 nodes,
12183 on tree, 880 best solution, best possible 749.53377 (3564.79
seconds) Cbc0010I After 38900 nodes, 12228 on tree, 880 best solution,
best possible 749.53377 (3574.20 seconds) Cbc0010I After 39000 nodes,
12269 on tree, 880 best solution, best possible 749.53377 (3584.21
seconds) Cbc0010I After 39100 nodes, 12315 on tree, 880 best solution,
best possible 749.53377 (3593.44 seconds) Cbc0010I After 39200 nodes,
12360 on tree, 880 best solution, best possible 749.53377 (3603.12
seconds) Cbc0010I After 39300 nodes, 12407 on tree, 880 best solution,
best possible 749.53377 (3612.69 seconds) Cbc0010I After 39400 nodes,
12450 on tree, 880 best solution, best possible 749.53377 (3623.39
seconds) Cbc0010I After 39500 nodes, 12490 on tree, 880 best solution,
best possible 749.53377 (3634.07 seconds) Cbc0038I Full problem 2890 rows
1451 columns, reduced to 1905 rows 802 columns - 1 fixed gives 1902, 800 -
still too large Cbc0010I After 39600 nodes, 12533 on tree, 880 best
solution, best possible 749.53377 (3643.89 seconds) Cbc0010I After 39700
nodes, 12575 on tree, 880 best solution, best possible 749.53377 (3655.23
seconds) Cbc0010I After 39800 nodes, 12623 on tree, 880 best solution,
best possible 749.53377 (3664.25 seconds) Cbc0010I After 39900 nodes,
12670 on tree, 880 best solution, best possible 749.53377 (3673.87
seconds) Cbc0010I After 40000 nodes, 12715 on tree, 880 best solution,
best possible 749.53377 (3681.33 seconds) Cbc0016I Integer solution of 878
found by strong branching after 13530063 iterations and 40069 nodes
(3683.62 seconds) Cbc0010I After 40100 nodes, 12651 on tree, 878 best
solution, best possible 749.53377 (3688.28 seconds) Cbc0010I After 40200
nodes, 12694 on tree, 878 best solution, best possible 749.53377 (3702.36
seconds) Cbc0010I After 40300 nodes, 12734 on tree, 878 best solution,
best possible 749.53377 (3713.43 seconds) Cbc0010I After 40400 nodes,
12774 on tree, 878 best solution, best possible 749.53377 (3724.82
seconds) Cbc0010I After 40500 nodes, 12817 on tree, 878 best solution,
best possible 749.53377 (3736.37 seconds) Cbc0010I After 40600 nodes,
12859 on tree, 878 best solution, best possible 749.53377 (3747.06
seconds) Cbc0010I After 40700 nodes, 12907 on tree, 878 best solution,
best possible 749.53377 (3756.49 seconds) Cbc0010I After 40800 nodes,
12952 on tree, 878 best solution, best possible 749.53377 (3767.08
seconds) Cbc0010I After 40900 nodes, 13001 on tree, 878 best solution,
best possible 749.53377 (3776.79 seconds) Cbc0010I After 41000 nodes,
13046 on tree, 878 best solution, best possible 749.53377 (3787.46
seconds) Cbc0010I After 41100 nodes, 13051 on tree, 878 best solution,
best possible 749.54718 (3794.05 seconds) Cbc0010I After 41200 nodes,
13050 on tree, 878 best solution, best possible 749.54718 (3800.56
```

```
seconds) Cbc0010I After 41300 nodes, 13047 on tree, 878 best solution,
best possible 749.54718 (3805.09 seconds) Cbc0010I After 41400 nodes,
13051 on tree, 878 best solution, best possible 749.54718 (3811.56
seconds) Cbc0010I After 41500 nodes, 13051 on tree, 878 best solution,
best possible 749.54718 (3817.33 seconds) Cbc0010I After 41600 nodes,
13054 on tree, 878 best solution, best possible 749.54718 (3822.68
seconds) Cbc0010I After 41700 nodes, 13052 on tree, 878 best solution,
best possible 749.54718 (3828.24 seconds) Cbc0010I After 41800 nodes,
13051 on tree, 878 best solution, best possible 749.54718 (3834.69
seconds) Cbc0010I After 41900 nodes, 13044 on tree, 878 best solution,
best possible 749.54718 (3838.99 seconds) Cbc0010I After 42000 nodes,
13029 on tree, 878 best solution, best possible 749.54718 (3841.82
seconds) Cbc0010I After 42100 nodes, 13078 on tree, 878 best solution,
best possible 749.54718 (3852.81 seconds) Cbc0010I After 42200 nodes,
13126 on tree, 878 best solution, best possible 749.54718 (3865.74
seconds) Cbc0010I After 42300 nodes, 13175 on tree, 878 best solution,
best possible 749.54718 (3874.16 seconds) Cbc0010I After 42400 nodes,
13219 on tree, 878 best solution, best possible 749.54718 (3885.07
seconds) Cbc0010I After 42500 nodes, 13265 on tree, 878 best solution,
best possible 749.54718 (3897.82 seconds) Cbc0010I After 42600 nodes,
13310 on tree, 878 best solution, best possible 749.54718 (3909.05
seconds) Cbc0010I After 42700 nodes, 13352 on tree, 878 best solution,
best possible 749.54718 (3920.17 seconds) Cbc0010I After 42800 nodes,
13395 on tree, 878 best solution, best possible 749.54718 (3931.87
seconds) Cbc0010I After 42900 nodes, 13438 on tree, 878 best solution,
best possible 749.54718 (3942.51 seconds) Cbc0010I After 43000 nodes,
13482 on tree, 878 best solution, best possible 749.54718 (3952.90
seconds) Cbc0010I After 43100 nodes, 13523 on tree, 878 best solution,
best possible 749.55451 (3964.29 seconds) Cbc0038I Full problem 2890 rows
1451 columns, reduced to 1819 rows 745 columns - 1 fixed gives 1816, 743 -
still too large Cbc0010I After 43200 nodes, 13569 on tree, 878 best
solution, best possible 749.55451 (3975.69 seconds) Cbc0010I After 43300
nodes, 13618 on tree, 878 best solution, best possible 749.55451 (3986.95
seconds) Cbc0010I After 43400 nodes, 13656 on tree, 878 best solution,
best possible 749.55451 (3997.08 seconds) Cbc0010I After 43500 nodes,
13705 on tree, 878 best solution, best possible 749.55451 (4007.01
seconds) Cbc0010I After 43600 nodes, 13753 on tree, 878 best solution,
best possible 749.55451 (4017.93 seconds) Cbc0010I After 43700 nodes,
13792 on tree, 878 best solution, best possible 749.55451 (4026.44
seconds) Cbc0010I After 43800 nodes, 13832 on tree, 878 best solution,
best possible 749.55451 (4033.71 seconds) Cbc0010I After 43900 nodes,
13870 on tree, 878 best solution, best possible 749.55451 (4040.20
seconds) Cbc0010I After 44000 nodes, 13909 on tree, 878 best solution,
best possible 749.55451 (4051.46 seconds) Cbc0010I After 44100 nodes,
13896 on tree, 878 best solution, best possible 749.55451 (4054.20
seconds) Cbc0010I After 44200 nodes, 13881 on tree, 878 best solution,
best possible 749.55451 (4057.33 seconds) Cbc0010I After 44300 nodes,
13881 on tree, 878 best solution, best possible 749.55451 (4059.53
seconds) Cbc0010I After 44400 nodes, 13871 on tree, 878 best solution,
best possible 749.55451 (4062.41 seconds) Cbc0010I After 44500 nodes,
13859 on tree, 878 best solution, best possible 749.55451 (4065.31
seconds) Cbc0010I After 44600 nodes, 13855 on tree, 878 best solution,
best possible 749.55451 (4067.76 seconds) Cbc0010I After 44700 nodes,
13853 on tree, 878 best solution, best possible 749.55451 (4070.76
seconds) Cbc0010I After 44800 nodes, 13853 on tree, 878 best solution,
best possible 749.55451 (4073.51 seconds) Cbc0010I After 44900 nodes,
13844 on tree, 878 best solution, best possible 749.55451 (4075.75
seconds) Cbc0010I After 45000 nodes, 13831 on tree, 878 best solution,
best possible 749.55451 (4079.06 seconds) Cbc0010I After 45100 nodes,
13841 on tree, 878 best solution, best possible 749.55451 (4085.08
seconds) Cbc0010I After 45200 nodes, 13841 on tree, 878 best solution,
best possible 749.55451 (4090.21 seconds) Cbc0010I After 45300 nodes,
13842 on tree, 878 best solution, best possible 749.55451 (4094.54
seconds) Cbc0010I After 45400 nodes, 13840 on tree, 878 best solution,
best possible 749.55451 (4098.55 seconds) Cbc0010I After 45500 nodes,
13846 on tree, 878 best solution, best possible 749.55451 (4103.05
seconds) Cbc0010I After 45600 nodes, 13848 on tree, 878 best solution,
best possible 749.55451 (4107.13 seconds) Cbc0010I After 45700 nodes,
```

```
13845 on tree, 878 best solution, best possible 749.55451 (4111.33
seconds) Cbc0010I After 45800 nodes, 13847 on tree, 878 best solution,
best possible 749.55451 (4115.79 seconds) Cbc0010I After 45900 nodes,
13845 on tree, 878 best solution, best possible 749.55451 (4118.87
seconds) Cbc0010I After 46000 nodes, 13844 on tree, 878 best solution,
best possible 749.55451 (4123.06 seconds) Cbc0010I After 46100 nodes,
13886 on tree, 878 best solution, best possible 749.55451 (4134.15
seconds) Cbc0010I After 46200 nodes, 13933 on tree, 878 best solution,
best possible 749.55451 (4145.90 seconds) Cbc0010I After 46300 nodes,
13974 on tree, 878 best solution, best possible 749.55451 (4155.66
seconds) Cbc0010I After 46400 nodes, 14020 on tree, 878 best solution,
best possible 749.55451 (4168.16 seconds) Cbc0010I After 46500 nodes,
14063 on tree, 878 best solution, best possible 749.55451 (4177.30
seconds) Cbc0010I After 46600 nodes, 14109 on tree, 878 best solution,
best possible 749.55451 (4188.01 seconds) Cbc0010I After 46700 nodes,
14153 on tree, 878 best solution, best possible 749.55451 (4197.81
seconds) Cbc0038I Full problem 2890 rows 1451 columns, reduced to 1744
rows 695 columns - 1 fixed gives 1741, 693 - still too large Cbc0010I
After 46800 nodes, 14196 on tree, 878 best solution, best possible
749.55451 (4207.68 seconds) Cbc0010I After 46900 nodes, 14241 on tree, 878
best solution, best possible 749.55451 (4218.06 seconds) Cbc0010I After
47000 nodes, 14285 on tree, 878 best solution, best possible 749.55451
(4228.36 seconds) Cbc0010I After 47100 nodes, 14326 on tree, 878 best
solution, best possible 749.55451 (4237.31 seconds) Cbc0010I After 47200
nodes, 14372 on tree, 878 best solution, best possible 749.55451 (4247.80
seconds) Cbc0010I After 47300 nodes, 14414 on tree, 878 best solution,
best possible 749.55451 (4257.45 seconds) Cbc0010I After 47400 nodes,
14464 on tree, 878 best solution, best possible 749.55451 (4268.40
seconds) Cbc0010I After 47500 nodes, 14505 on tree, 878 best solution,
best possible 749.55451 (4277.66 seconds) Cbc0010I After 47600 nodes,
14553 on tree, 878 best solution, best possible 749.55451 (4288.76
seconds) Cbc0010I After 47700 nodes, 14598 on tree, 878 best solution,
best possible 749.55451 (4298.49 seconds) Cbc0010I After 47800 nodes,
14637 on tree, 878 best solution, best possible 749.55451 (4308.45
seconds) Cbc0010I After 47900 nodes, 14682 on tree, 878 best solution,
best possible 749.55451 (4318.82 seconds) Cbc0010I After 48000 nodes,
14720 on tree, 878 best solution, best possible 749.55451 (4329.10
seconds) Cbc0010I After 48100 nodes, 14717 on tree, 878 best solution,
best possible 749.55451 (4332.06 seconds) Cbc0010I After 48200 nodes,
14708 on tree, 878 best solution, best possible 749.55451 (4334.65
seconds) Cbc0010I After 48300 nodes, 14703 on tree, 878 best solution,
best possible 749.55451 (4337.34 seconds) Cbc0010I After 48400 nodes,
14703 on tree, 878 best solution, best possible 749.55451 (4339.13
seconds) Cbc0004I Integer solution of 877 found after 16037696 iterations
and 48455 nodes (4340.62 seconds) Cbc0010I After 48500 nodes, 14702 on
tree, 877 best solution, best possible 749.55451 (4343.84 seconds)
Cbc0010I After 48600 nodes, 14745 on tree, 877 best solution, best
possible 749.55451 (4351.93 seconds) Cbc0010I After 48700 nodes, 14788 on
tree, 877 best solution, best possible 749.55451 (4360.56 seconds)
Cbc0010I After 48800 nodes, 14832 on tree, 877 best solution, best
possible 749.55451 (4368.84 seconds) Cbc0010I After 48900 nodes, 14876 on
tree, 877 best solution, best possible 749.55451 (4379.21 seconds)
Cbc0010I After 49000 nodes, 14919 on tree, 877 best solution, best
possible 749.55451 (4388.57 seconds) Cbc0010I After 49100 nodes, 14927 on
tree, 877 best solution, best possible 749.58541 (4394.67 seconds)
Cbc0010I After 49200 nodes, 14929 on tree, 877 best solution, best
possible 749.58541 (4400.31 seconds) Cbc0010I After 49300 nodes, 14929 on
tree, 877 best solution, best possible 749.58541 (4405.30 seconds)
Cbc0010I After 49400 nodes, 14921 on tree, 877 best solution, best
possible 749.58541 (4410.88 seconds) Cbc0010I After 49500 nodes, 14915 on
tree, 877 best solution, best possible 749.58541 (4415.32 seconds)
Cbc0010I After 49600 nodes, 14915 on tree, 877 best solution, best
possible 749.58541 (4418.04 seconds) Cbc0010I After 49700 nodes, 14906 on
tree, 877 best solution, best possible 749.58541 (4420.82 seconds)
Cbc0010I After 49800 nodes, 14915 on tree, 877 best solution, best
possible 749.58541 (4423.36 seconds) Cbc0010I After 49900 nodes, 14915 on
tree, 877 best solution, best possible 749.58541 (4425.86 seconds)
Cbc0010I After 50000 nodes, 14895 on tree, 877 best solution, best
possible 749.58541 (4428.47 seconds) Cbc0010I After 50100 nodes, 14944 on
```

```
tree, 877 best solution, best possible 749.58541 (4442.85 seconds)
Cbc0010I After 50200 nodes, 14989 on tree, 877 best solution, best
possible 749.58541 (4453.77 seconds) Cbc0010I After 50300 nodes, 15030 on
tree, 877 best solution, best possible 749.58541 (4465.61 seconds)
Cbc0010I After 50400 nodes, 15076 on tree, 877 best solution, best
possible 749.58541 (4476.21 seconds) Cbc0010I After 50500 nodes, 15122 on
tree, 877 best solution, best possible 749.58541 (4487.39 seconds)
Cbc0010I After 50600 nodes, 15166 on tree, 877 best solution, best
possible 749.58541 (4497.96 seconds) Cbc0010I After 50700 nodes, 15212 on
tree, 877 best solution, best possible 749.58541 (4508.91 seconds)
Cbc0010I After 50800 nodes, 15259 on tree, 877 best solution, best
possible 749.58541 (4520.39 seconds) Cbc0010I After 50900 nodes, 15306 on
tree, 877 best solution, best possible 749.58541 (4532.47 seconds)
Cbc0010I After 51000 nodes, 15350 on tree, 877 best solution, best
possible 749.58541 (4543.58 seconds) Cbc0010I After 51100 nodes, 15393 on
tree, 877 best solution, best possible 749.58541 (4552.54 seconds)
Cbc0010I After 51200 nodes, 15440 on tree, 877 best solution, best
possible 749.58541 (4561.98 seconds) Cbc0010I After 51300 nodes, 15489 on
tree, 877 best solution, best possible 749.58541 (4572.32 seconds)
Cbc0010I After 51400 nodes, 15525 on tree, 877 best solution, best
possible 749.58541 (4582.14 seconds) Cbc0010I After 51500 nodes, 15574 on
tree, 877 best solution, best possible 749.58541 (4593.31 seconds)
Cbc0010I After 51600 nodes, 15620 on tree, 877 best solution, best
possible 749.58541 (4604.43 seconds) Cbc0010I After 51700 nodes, 15669 on
tree, 877 best solution, best possible 749.58541 (4616.53 seconds)
Cbc0010I After 51800 nodes, 15713 on tree, 877 best solution, best
possible 749.58541 (4627.70 seconds) Cbc0010I After 51900 nodes, 15759 on
tree, 877 best solution, best possible 749.58541 (4637.97 seconds)
Cbc0010I After 52000 nodes, 15798 on tree, 877 best solution, best
possible 749.58541 (4649.50 seconds) Cbc0010I After 52100 nodes, 15791 on
tree, 877 best solution, best possible 749.58541 (4652.18 seconds)
Cbc0010I After 52200 nodes, 15789 on tree, 877 best solution, best
possible 749.58541 (4655.06 seconds) Cbc0010I After 52300 nodes, 15784 on
tree, 877 best solution, best possible 749.58541 (4657.66 seconds)
Cbc0010I After 52400 nodes, 15768 on tree, 877 best solution, best
possible 749.58541 (4660.50 seconds) Cbc0010I After 52500 nodes, 15767 on
tree, 877 best solution, best possible 749.58541 (4663.73 seconds)
Cbc0010I After 52600 nodes, 15760 on tree, 877 best solution, best
possible 749.58541 (4666.21 seconds) Cbc0010I After 52700 nodes, 15761 on
tree, 877 best solution, best possible 749.58541 (4669.31 seconds)
Cbc0010I After 52800 nodes, 15750 on tree, 877 best solution, best
possible 749.58541 (4672.09 seconds) Cbc0010I After 52900 nodes, 15747 on
tree, 877 best solution, best possible 749.58541 (4674.47 seconds)
Cbc0010I After 53000 nodes, 15742 on tree, 877 best solution, best
possible 749.58541 (4677.10 seconds) Cbc0010I After 53100 nodes, 15758 on
tree, 877 best solution, best possible 749.58541 (4681.99 seconds)
Cbc0010I After 53200 nodes, 15759 on tree, 877 best solution, best
possible 749.58541 (4685.49 seconds) Cbc0010I After 53300 nodes, 15755 on
tree, 877 best solution, best possible 749.58541 (4689.72 seconds)
Cbc0010I After 53400 nodes, 15763 on tree, 877 best solution, best
possible 749.58541 (4693.69 seconds) Cbc0010I After 53500 nodes, 15754 on
tree, 877 best solution, best possible 749.58541 (4697.89 seconds)
Cbc0010I After 53600 nodes, 15759 on tree, 877 best solution, best
possible 749.58541 (4702.55 seconds) Cbc0010I After 53700 nodes, 15754 on
tree, 877 best solution, best possible 749.58541 (4707.12 seconds)
Cbc0010I After 53800 nodes, 15760 on tree, 877 best solution, best
possible 749.58541 (4711.59 seconds) Cbc0010I After 53900 nodes, 15757 on
tree, 877 best solution, best possible 749.58541 (4716.08 seconds)
Cbc0010I After 54000 nodes, 15751 on tree, 877 best solution, best
possible 749.58541 (4720.77 seconds) Cbc0010I After 54100 nodes, 15794 on
tree, 877 best solution, best possible 749.58541 (4730.99 seconds)
Cbc0010I After 54200 nodes, 15842 on tree, 877 best solution, best
possible 749.58541 (4741.95 seconds) Cbc0010I After 54300 nodes, 15889 on
tree, 877 best solution, best possible 749.58541 (4749.30 seconds)
Cbc0010I After 54400 nodes, 15934 on tree, 877 best solution, best
possible 749.58541 (4758.42 seconds) Cbc0010I After 54500 nodes, 15978 on
tree, 877 best solution, best possible 749.58541 (4768.70 seconds)
Cbc0010I After 54600 nodes, 16017 on tree, 877 best solution, best
```

```
possible 749.58541 (4779.28 seconds) Cbc0010I After 54700 nodes, 16059 on
tree, 877 best solution, best possible 749.58541 (4789.44 seconds)
Cbc0010I After 54800 nodes, 16102 on tree, 877 best solution, best
possible 749.58541 (4801.07 seconds) Cbc0010I After 54900 nodes, 16148 on
tree, 877 best solution, best possible 749.58541 (4812.60 seconds)
Cbc0010I After 55000 nodes, 16194 on tree, 877 best solution, best
possible 749.58541 (4821.03 seconds) Cbc0010I After 55100 nodes, 16237 on
tree, 877 best solution, best possible 749.58541 (4831.55 seconds)
Cbc0010I After 55200 nodes, 16280 on tree, 877 best solution, best
possible 749.58541 (4842.14 seconds) Cbc0010I After 55300 nodes, 16326 on
tree, 877 best solution, best possible 749.58541 (4851.44 seconds)
Cbc0010I After 55400 nodes, 16370 on tree, 877 best solution, best
possible 749.58541 (4860.75 seconds) Cbc0010I After 55500 nodes, 16414 on
tree, 877 best solution, best possible 749.58541 (4870.45 seconds)
Cbc0010I After 55600 nodes, 16458 on tree, 877 best solution, best
possible 749.58541 (4882.06 seconds) Cbc0010I After 55700 nodes, 16506 on
tree, 877 best solution, best possible 749.58541 (4893.20 seconds)
Cbc0010I After 55800 nodes, 16548 on tree, 877 best solution, best
possible 749.58541 (4905.23 seconds) Cbc0010I After 55900 nodes, 16592 on
tree, 877 best solution, best possible 749.58541 (4914.58 seconds)
Cbc0010I After 56000 nodes, 16635 on tree, 877 best solution, best
possible 749.58541 (4924.57 seconds) Cbc0004I Integer solution of 875
found after 18383734 iterations and 56041 nodes (4925.94 seconds) Cbc0010I
After 56100 nodes, 16500 on tree, 875 best solution, best possible
749.58541 (4931.64 seconds) Cbc0010I After 56200 nodes, 16544 on tree, 875
best solution, best possible 749.58541 (4940.87 seconds) Cbc0010I After
56300 nodes, 16580 on tree, 875 best solution, best possible 749.58541
(4947.95 seconds) Cbc0010I After 56400 nodes, 16617 on tree, 875 best
solution, best possible 749.58541 (4954.85 seconds) Cbc0010I After 56500
nodes, 16660 on tree, 875 best solution, best possible 749.58541 (4964.29
seconds) Cbc0010I After 56600 nodes, 16703 on tree, 875 best solution,
best possible 749.58541 (4974.87 seconds) Cbc0010I After 56700 nodes,
16748 on tree, 875 best solution, best possible 749.58541 (4985.45
seconds) Cbc0010I After 56800 nodes, 16794 on tree, 875 best solution,
best possible 749.58541 (4994.24 seconds) Cbc0010I After 56900 nodes,
16835 on tree, 875 best solution, best possible 749.58541 (5003.61
seconds) Cbc0010I After 57000 nodes, 16879 on tree, 875 best solution,
best possible 749.58541 (5012.28 seconds) Cbc0010I After 57100 nodes,
16888 on tree, 875 best solution, best possible 749.58891 (5018.32
seconds) Cbc0010I After 57200 nodes, 16892 on tree, 875 best solution,
best possible 749.58891 (5024.35 seconds) Cbc0010I After 57300 nodes,
16884 on tree, 875 best solution, best possible 749.58891 (5030.04
seconds) Cbc0010I After 57400 nodes, 16881 on tree, 875 best solution,
best possible 749.58891 (5035.48 seconds) Cbc0010I After 57500 nodes,
16883 on tree, 875 best solution, best possible 749.58891 (5041.15
seconds) Cbc0010I After 57600 nodes, 16888 on tree, 875 best solution,
best possible 749.58891 (5045.87 seconds) Cbc0010I After 57700 nodes,
16883 on tree, 875 best solution, best possible 749.58891 (5050.62
seconds) Cbc0010I After 57800 nodes, 16880 on tree, 875 best solution,
best possible 749.58891 (5056.48 seconds) Cbc0010I After 57900 nodes,
16865 on tree, 875 best solution, best possible 749.58891 (5059.06
seconds) Cbc0010I After 58000 nodes, 16858 on tree, 875 best solution,
best possible 749.58891 (5061.61 seconds) Cbc0010I After 58100 nodes,
16904 on tree, 875 best solution, best possible 749.58891 (5075.12
seconds) Cbc0010I After 58200 nodes, 16950 on tree, 875 best solution,
best possible 749.58891 (5087.17 seconds) Cbc0010I After 58300 nodes,
16996 on tree, 875 best solution, best possible 749.58891 (5100.72
seconds) Cbc0010I After 58400 nodes, 17043 on tree, 875 best solution,
best possible 749.58891 (5112.85 seconds) Cbc0010I After 58500 nodes,
17090 on tree, 875 best solution, best possible 749.58891 (5123.67
seconds) Cbc0010I After 58600 nodes, 17140 on tree, 875 best solution,
best possible 749.58891 (5135.08 seconds) Cbc0010I After 58700 nodes,
17184 on tree, 875 best solution, best possible 749.58891 (5146.74
seconds) Cbc0010I After 58800 nodes, 17232 on tree, 875 best solution,
best possible 749.58891 (5160.10 seconds) Cbc0010I After 58900 nodes,
17279 on tree, 875 best solution, best possible 749.58891 (5171.58
seconds) Cbc0010I After 59000 nodes, 17323 on tree, 875 best solution,
best possible 749.58891 (5181.45 seconds) Cbc0010I After 59100 nodes,
17366 on tree, 875 best solution, best possible 749.58891 (5191.83
```

17500 on tree, 875 best solution, best possible 749.58891 (5191.65
seconds) Cbc0010I After 59200 nodes, 17413 on tree, 875 best solution,
best possible 749.58891 (5203.30 seconds) Cbc0010I After 59300 nodes,
17449 on tree, 875 best solution, best possible 749.58891 (5212.93
seconds) Cbc0010I After 59400 nodes, 17492 on tree, 875 best solution,
best possible 749.58891 (5224.36 seconds) Cbc0010I After 59500 nodes,
17540 on tree, 875 best solution, best possible 749.58891 (5235.82
seconds) Cbc0010I After 59600 nodes, 17584 on tree, 875 best solution,
best possible 749.58891 (5246.50 seconds) Cbc0010I After 59700 nodes,
17624 on tree, 875 best solution, best possible 749.58891 (5254.69
seconds) Cbc0010I After 59800 nodes, 17667 on tree, 875 best solution,
best possible 749.58891 (5266.18 seconds) Cbc0010I After 59900 nodes,
17714 on tree, 875 best solution, best possible 749.58891 (5277.38
seconds) Cbc0010I After 60000 nodes, 17756 on tree, 875 best solution,
best possible 749.58891 (5290.00 seconds) Cbc0010I After 60100 nodes,
17754 on tree, 875 best solution, best possible 749.58891 (5293.24
seconds) Cbc0010I After 60200 nodes, 17747 on tree, 875 best solution,
best possible 749.58891 (5296.48 seconds) Cbc0010I After 60300 nodes,
17734 on tree, 875 best solution, best possible 749.58891 (5299.60
seconds) Cbc0010I After 60400 nodes, 17727 on tree, 875 best solution,
best possible 749.58891 (5302.38 seconds) Cbc0010I After 60500 nodes,
17718 on tree, 875 best solution, best possible 749.58891 (5305.46
seconds) Cbc0010I After 60600 nodes, 17714 on tree, 875 best solution,
best possible 749.58891 (5308.11 seconds) Cbc0010I After 60700 nodes,
17710 on tree, 875 best solution, best possible 749.58891 (5311.21
seconds) Cbc0010I After 60800 nodes, 17703 on tree, 875 best solution,
best possible 749.58891 (5314.39 seconds) Cbc0010I After 60900 nodes,
17705 on tree, 875 best solution, best possible 749.58891 (5317.53
seconds) Cbc0010I After 61000 nodes, 17709 on tree, 875 best solution,
best possible 749.58891 (5320.11 seconds) Cbc0010I After 61100 nodes,
17717 on tree, 875 best solution, best possible 749.58891 (5325.80
seconds) Cbc0010I After 61200 nodes, 17722 on tree, 875 best solution,
best possible 749.58891 (5330.05 seconds) Cbc0010I After 61300 nodes,
17724 on tree, 875 best solution, best possible 749.58891 (5334.63
seconds) Cbc0010I After 61400 nodes, 17722 on tree, 875 best solution,
best possible 749.58891 (5338.17 seconds) Cbc0010I After 61500 nodes,
17720 on tree, 875 best solution, best possible 749.58891 (5342.04
seconds) Cbc0010I After 61600 nodes, 17719 on tree, 875 best solution,
best possible 749.58891 (5345.65 seconds) Cbc0010I After 61700 nodes,
17731 on tree, 875 best solution, best possible 749.58891 (5349.44
seconds) Cbc0010I After 61800 nodes, 17722 on tree, 875 best solution,
best possible 749.58891 (5352.85 seconds) Cbc0010I After 61900 nodes,
17727 on tree, 875 best solution, best possible 749.58891 (5356.62
seconds) Cbc0010I After 62000 nodes, 17721 on tree, 875 best solution,
best possible 749.58891 (5360.21 seconds) Cbc0010I After 62100 nodes,
17762 on tree, 875 best solution, best possible 749.62686 (5372.94
seconds) Cbc0010I After 62200 nodes, 17805 on tree, 875 best solution,
best possible 749.62686 (5382.87 seconds) Cbc0010I After 62300 nodes,
17844 on tree, 875 best solution, best possible 749.62686 (5392.61
seconds) Cbc0010I After 62400 nodes, 17889 on tree, 875 best solution,
best possible 749.62686 (5402.40 seconds) Cbc0010I After 62500 nodes,
17933 on tree, 875 best solution, best possible 749.62686 (5412.83
seconds) Cbc0010I After 62600 nodes, 17979 on tree, 875 best solution,
best possible 749.62686 (5424.02 seconds) Cbc0010I After 62700 nodes,
18021 on tree, 875 best solution, best possible 749.62686 (5435.61
seconds) Cbc0010I After 62800 nodes, 18062 on tree, 875 best solution,
best possible 749.62686 (5445.69 seconds) Cbc0010I After 62900 nodes,
18102 on tree, 875 best solution, best possible 749.62686 (5455.78
seconds) Cbc0010I After 63000 nodes, 18144 on tree, 875 best solution,
best possible 749.62686 (5465.91 seconds) Cbc0010I After 63100 nodes,
18188 on tree, 875 best solution, best possible 749.62686 (5478.82
seconds) Cbc0010I After 63200 nodes, 18233 on tree, 875 best solution,
best possible 749.62686 (5489.61 seconds) Cbc0010I After 63300 nodes,
18272 on tree, 875 best solution, best possible 749.62686 (5498.73
seconds) Cbc0010I After 63400 nodes, 18311 on tree, 875 best solution,
best possible 749.62686 (5508.05 seconds) Cbc0010I After 63500 nodes,
18353 on tree, 875 best solution, best possible 749.62686 (5518.96
seconds) Cbc0010I After 63600 nodes, 18398 on tree, 875 best solution,
best possible 749.62686 (5527.29 seconds) Cbc0010I After 63700 nodes,

```
18444 on tree, 875 best solution, best possible 749.62686 (5537.30
seconds) Cbc0010I After 63800 nodes, 18483 on tree, 875 best solution,
best possible 749.62686 (5548.20 seconds) Cbc0010I After 63900 nodes,
18526 on tree, 875 best solution, best possible 749.62686 (5559.13
seconds) Cbc0010I After 64000 nodes, 18569 on tree, 875 best solution,
best possible 749.62686 (5570.21 seconds) Cbc0010I After 64100 nodes,
18568 on tree, 875 best solution, best possible 749.62686 (5572.49
seconds) Cbc0010I After 64200 nodes, 18564 on tree, 875 best solution,
best possible 749.62686 (5574.57 seconds) Cbc0010I After 64300 nodes,
18558 on tree, 875 best solution, best possible 749.62686 (5577.24
seconds) Cbc0012I Integer solution of 872 found by rounding after 20954446
iterations and 64393 nodes (5579.76 seconds) Cbc0010I After 64400 nodes,
18424 on tree, 872 best solution, best possible 749.62686 (5580.94
seconds) Cbc0010I After 64500 nodes, 18460 on tree, 872 best solution,
best possible 749.62686 (5588.32 seconds) Cbc0010I After 64600 nodes,
18492 on tree, 872 best solution, best possible 749.62686 (5594.51
seconds) Cbc0010I After 64700 nodes, 18533 on tree, 872 best solution,
best possible 749.62686 (5603.63 seconds) Cbc0038I Full problem 2890 rows
1451 columns, reduced to 1762 rows 707 columns - 1 fixed gives 1759, 705 -
still too large Cbc0010I After 64800 nodes, 18572 on tree, 872 best
solution, best possible 749.62686 (5612.80 seconds) Cbc0010I After 64900
nodes, 18614 on tree, 872 best solution, best possible 749.62686 (5621.86
seconds) Cbc0010I After 65000 nodes, 18648 on tree, 872 best solution,
best possible 749.62686 (5631.52 seconds) Cbc0010I After 65100 nodes,
18664 on tree, 872 best solution, best possible 749.62686 (5639.07
seconds) Cbc0010I After 65200 nodes, 18657 on tree, 872 best solution,
best possible 749.62686 (5643.54 seconds) Cbc0010I After 65300 nodes,
18666 on tree, 872 best solution, best possible 749.62686 (5648.88
seconds) Cbc0010I After 65400 nodes, 18663 on tree, 872 best solution,
best possible 749.62686 (5653.82 seconds) Cbc0010I After 65500 nodes,
18658 on tree, 872 best solution, best possible 749.62686 (5658.41
seconds) Cbc0010I After 65600 nodes, 18664 on tree, 872 best solution,
best possible 749.62686 (5662.65 seconds) Cbc0010I After 65700 nodes,
18657 on tree, 872 best solution, best possible 749.62686 (5667.07
seconds) Cbc0010I After 65800 nodes, 18662 on tree, 872 best solution,
best possible 749.62686 (5671.48 seconds) Cbc0010I After 65900 nodes,
18662 on tree, 872 best solution, best possible 749.62686 (5675.86
seconds) Cbc0010I After 66000 nodes, 18662 on tree, 872 best solution,
best possible 749.62686 (5678.86 seconds) Cbc0010I After 66100 nodes,
18708 on tree, 872 best solution, best possible 749.62686 (5692.09
seconds) Cbc0010I After 66200 nodes, 18754 on tree, 872 best solution,
best possible 749.62686 (5701.04 seconds) Cbc0010I After 66300 nodes,
18800 on tree, 872 best solution, best possible 749.62686 (5712.02
seconds) Cbc0010I After 66400 nodes, 18846 on tree, 872 best solution,
best possible 749.62686 (5725.19 seconds) Cbc0010I After 66500 nodes,
18894 on tree, 872 best solution, best possible 749.62686 (5736.42
seconds) Cbc0010I After 66600 nodes, 18939 on tree, 872 best solution,
best possible 749.62686 (5747.79 seconds) Cbc0010I After 66700 nodes,
18984 on tree, 872 best solution, best possible 749.62686 (5760.36
seconds) Cbc0010I After 66800 nodes, 19031 on tree, 872 best solution,
best possible 749.62686 (5771.37 seconds) Cbc0010I After 66900 nodes,
19079 on tree, 872 best solution, best possible 749.62686 (5782.29
seconds) Cbc0010I After 67000 nodes, 19125 on tree, 872 best solution,
best possible 749.62686 (5793.36 seconds) Cbc0010I After 67100 nodes,
19171 on tree, 872 best solution, best possible 749.62686 (5805.79
seconds) Cbc0010I After 67200 nodes, 19215 on tree, 872 best solution,
best possible 749.62686 (5817.07 seconds) Cbc0010I After 67300 nodes,
19264 on tree, 872 best solution, best possible 749.62686 (5829.20
seconds) Cbc0010I After 67400 nodes, 19306 on tree, 872 best solution,
best possible 749.62686 (5841.42 seconds) Cbc0010I After 67500 nodes,
19352 on tree, 872 best solution, best possible 749.62686 (5853.67
seconds) Cbc0010I After 67600 nodes, 19397 on tree, 872 best solution,
best possible 749.62686 (5864.97 seconds) Cbc0010I After 67700 nodes,
19441 on tree, 872 best solution, best possible 749.62686 (5875.73
seconds) Cbc0010I After 67800 nodes, 19484 on tree, 872 best solution,
best possible 749.62686 (5883.88 seconds) Cbc0010I After 67900 nodes,
19531 on tree, 872 best solution, best possible 749.62686 (5892.81
seconds) Cbc0010I After 68000 nodes, 19573 on tree, 872 best solution,
best possible 749.62686 (5905.03 seconds) Cbc0010I After 68100 nodes,
```

```
best possible 749.62686 (5905.03 seconds) Cbc0010I After 68100 nodes,
19561 on tree, 872 best solution, best possible 749.62686 (5907.70
seconds) Cbc0010I After 68200 nodes, 19558 on tree, 872 best solution,
best possible 749.62686 (5910.96 seconds) Cbc0010I After 68300 nodes,
19556 on tree, 872 best solution, best possible 749.62686 (5914.06
seconds) Cbc0010I After 68400 nodes, 19551 on tree, 872 best solution,
best possible 749.62686 (5916.66 seconds) Cbc0010I After 68500 nodes,
19549 on tree, 872 best solution, best possible 749.62686 (5919.99
seconds) Cbc0010I After 68600 nodes, 19543 on tree, 872 best solution,
best possible 749.62686 (5922.82 seconds) Cbc0010I After 68700 nodes,
19540 on tree, 872 best solution, best possible 749.62686 (5925.51
seconds) Cbc0010I After 68800 nodes, 19527 on tree, 872 best solution,
best possible 749.62686 (5928.41 seconds) Cbc0010I After 68900 nodes,
19527 on tree, 872 best solution, best possible 749.62686 (5931.62
seconds) Cbc0010I After 69000 nodes, 19523 on tree, 872 best solution,
best possible 749.62686 (5934.00 seconds) Cbc0010I After 69100 nodes,
19532 on tree, 872 best solution, best possible 749.62893 (5942.49
seconds) Cbc0010I After 69200 nodes, 19532 on tree, 872 best solution,
best possible 749.62893 (5947.31 seconds) Cbc0010I After 69300 nodes,
19532 on tree, 872 best solution, best possible 749.62893 (5952.67
seconds) Cbc0010I After 69400 nodes, 19531 on tree, 872 best solution,
best possible 749.62893 (5958.06 seconds) Cbc0010I After 69500 nodes,
19533 on tree, 872 best solution, best possible 749.62893 (5962.65
seconds) Cbc0010I After 69600 nodes, 19534 on tree, 872 best solution,
best possible 749.62893 (5968.12 seconds) Cbc0010I After 69700 nodes,
19532 on tree, 872 best solution, best possible 749.62893 (5972.97
seconds) Cbc0010I After 69800 nodes, 19530 on tree, 872 best solution,
best possible 749.62893 (5978.25 seconds) Cbc0010I After 69900 nodes,
19533 on tree, 872 best solution, best possible 749.62893 (5984.85
seconds) Cbc0010I After 70000 nodes, 19534 on tree, 872 best solution,
best possible 749.62893 (5990.54 seconds) Cbc0010I After 70100 nodes,
19574 on tree, 872 best solution, best possible 749.62893 (6001.42
seconds) Cbc0010I After 70200 nodes, 19622 on tree, 872 best solution,
best possible 749.62893 (6012.35 seconds) Cbc0010I After 70300 nodes,
19666 on tree, 872 best solution, best possible 749.62893 (6024.42
seconds) Cbc0010I After 70400 nodes, 19707 on tree, 872 best solution,
best possible 749.62893 (6034.55 seconds) Cbc0010I After 70500 nodes,
19753 on tree, 872 best solution, best possible 749.62893 (6044.82
seconds) Cbc0010I After 70600 nodes, 19799 on tree, 872 best solution,
best possible 749.62893 (6058.24 seconds) Cbc0010I After 70700 nodes,
19848 on tree, 872 best solution, best possible 749.62893 (6071.73
seconds) Cbc0010I After 70800 nodes, 19890 on tree, 872 best solution,
best possible 749.62893 (6081.86 seconds) Cbc0010I After 70900 nodes,
19934 on tree, 872 best solution, best possible 749.62893 (6091.38
seconds) Cbc0010I After 71000 nodes, 19974 on tree, 872 best solution,
best possible 749.62893 (6101.67 seconds) Cbc0010I After 71100 nodes,
20016 on tree, 872 best solution, best possible 749.62893 (6113.36
seconds) Cbc0010I After 71200 nodes, 20059 on tree, 872 best solution,
best possible 749.62893 (6122.59 seconds) Cbc0010I After 71300 nodes,
20100 on tree, 872 best solution, best possible 749.62893 (6132.14
seconds) Cbc0010I After 71400 nodes, 20147 on tree, 872 best solution,
best possible 749.62893 (6141.35 seconds) Cbc0010I After 71500 nodes,
20180 on tree, 872 best solution, best possible 749.62893 (6149.11
seconds) Cbc0010I After 71600 nodes, 20224 on tree, 872 best solution,
best possible 749.62893 (6158.50 seconds) Cbc0010I After 71700 nodes,
20267 on tree, 872 best solution, best possible 749.62893 (6169.64
seconds) Cbc0010I After 71800 nodes, 20313 on tree, 872 best solution,
best possible 749.62893 (6180.27 seconds) Cbc0010I After 71900 nodes,
20356 on tree, 872 best solution, best possible 749.62893 (6190.17
seconds) Cbc0010I After 72000 nodes, 20395 on tree, 872 best solution,
best possible 749.62893 (6200.47 seconds) Cbc0010I After 72100 nodes,
20386 on tree, 872 best solution, best possible 749.62893 (6203.26
seconds) Cbc0010I After 72200 nodes, 20378 on tree, 872 best solution,
best possible 749.62893 (6206.41 seconds) Cbc0010I After 72300 nodes,
20370 on tree, 872 best solution, best possible 749.62893 (6209.30
seconds) Cbc0010I After 72400 nodes, 20374 on tree, 872 best solution,
best possible 749.62893 (6212.00 seconds) Cbc0010I After 72500 nodes,
20363 on tree, 872 best solution, best possible 749.62893 (6214.67
seconds) Cbc0010I After 72600 nodes, 20353 on tree, 872 best solution,
```

```
best possible 749.62893 (6217.75 seconds) Cbc0010I After 72700 nodes,
20344 on tree, 872 best solution, best possible 749.62893 (6220.77
seconds) Cbc0010I After 72800 nodes, 20347 on tree, 872 best solution,
best possible 749.62893 (6223.93 seconds) Cbc0010I After 72900 nodes,
20347 on tree, 872 best solution, best possible 749.62893 (6227.14
seconds) Cbc0010I After 73000 nodes, 20341 on tree, 872 best solution,
best possible 749.62893 (6229.92 seconds) Cbc0010I After 73100 nodes,
20359 on tree, 872 best solution, best possible 749.62893 (6234.67
seconds) Cbc0010I After 73200 nodes, 20359 on tree, 872 best solution,
best possible 749.62893 (6238.17 seconds) Cbc0010I After 73300 nodes,
20352 on tree, 872 best solution, best possible 749.62893 (6242.46
seconds) Cbc0010I After 73400 nodes, 20352 on tree, 872 best solution,
best possible 749.62893 (6246.78 seconds) Cbc0010I After 73500 nodes,
20349 on tree, 872 best solution, best possible 749.62893 (6250.88
seconds) Cbc0010I After 73600 nodes, 20356 on tree, 872 best solution,
best possible 749.62893 (6254.08 seconds) Cbc0010I After 73700 nodes,
20358 on tree, 872 best solution, best possible 749.62893 (6257.64
seconds) Cbc0010I After 73800 nodes, 20357 on tree, 872 best solution,
best possible 749.62893 (6261.48 seconds) Cbc0010I After 73900 nodes,
20363 on tree, 872 best solution, best possible 749.62893 (6265.59
seconds) Cbc0010I After 74000 nodes, 20364 on tree, 872 best solution,
best possible 749.62893 (6268.55 seconds) Cbc0010I After 74100 nodes,
20407 on tree, 872 best solution, best possible 749.62893 (6280.98
seconds) Cbc0010I After 74200 nodes, 20447 on tree, 872 best solution,
best possible 749.62893 (6290.71 seconds) Cbc0010I After 74300 nodes,
20489 on tree, 872 best solution, best possible 749.62893 (6300.27
seconds) Cbc0010I After 74400 nodes, 20534 on tree, 872 best solution,
best possible 749.62893 (6311.80 seconds) Cbc0010I After 74500 nodes,
20574 on tree, 872 best solution, best possible 749.62893 (6322.35
seconds) Cbc0010I After 74600 nodes, 20614 on tree, 872 best solution,
best possible 749.62893 (6332.54 seconds) Cbc0010I After 74700 nodes,
20656 on tree, 872 best solution, best possible 749.62893 (6342.63
seconds) Cbc0010I After 74800 nodes, 20702 on tree, 872 best solution,
best possible 749.62893 (6353.84 seconds) Cbc0010I After 74900 nodes,
20746 on tree, 872 best solution, best possible 749.62893 (6363.31
seconds) Cbc0010I After 75000 nodes, 20781 on tree, 872 best solution,
best possible 749.62893 (6371.71 seconds) Cbc0010I After 75100 nodes,
20827 on tree, 872 best solution, best possible 749.62893 (6384.42
seconds) Cbc0010I After 75200 nodes, 20867 on tree, 872 best solution,
best possible 749.62893 (6394.62 seconds) Cbc0010I After 75300 nodes,
20906 on tree, 872 best solution, best possible 749.62893 (6404.39
seconds) Cbc0010I After 75400 nodes, 20949 on tree, 872 best solution,
best possible 749.62893 (6415.95 seconds) Cbc0010I After 75500 nodes,
20991 on tree, 872 best solution, best possible 749.62893 (6425.46
seconds) Cbc0010I After 75600 nodes, 21037 on tree, 872 best solution,
best possible 749.62893 (6437.76 seconds) Cbc0010I After 75700 nodes,
21077 on tree, 872 best solution, best possible 749.62893 (6449.43
seconds) Cbc0010I After 75800 nodes, 21123 on tree, 872 best solution,
best possible 749.62893 (6458.35 seconds) Cbc0010I After 75900 nodes,
21164 on tree, 872 best solution, best possible 749.62893 (6468.15
seconds) Cbc0010I After 76000 nodes, 21207 on tree, 872 best solution,
best possible 749.62893 (6475.11 seconds) Cbc0010I After 76100 nodes,
21208 on tree, 872 best solution, best possible 749.62893 (6479.17
seconds) Cbc0010I After 76200 nodes, 21215 on tree, 872 best solution,
best possible 749.62893 (6483.07 seconds) Cbc0010I After 76300 nodes,
21209 on tree, 872 best solution, best possible 749.62893 (6486.12
seconds) Cbc0010I After 76400 nodes, 21210 on tree, 872 best solution,
best possible 749.62893 (6488.66 seconds) Cbc0010I After 76500 nodes,
21203 on tree, 872 best solution, best possible 749.62893 (6491.68
seconds) Cbc0010I After 76600 nodes, 21204 on tree, 872 best solution,
best possible 749.62893 (6494.87 seconds) Cbc0010I After 76700 nodes,
21192 on tree, 872 best solution, best possible 749.62893 (6497.59
seconds) Cbc0010I After 76800 nodes, 21192 on tree, 872 best solution,
best possible 749.62893 (6500.64 seconds) Cbc0010I After 76900 nodes,
21191 on tree, 872 best solution, best possible 749.62893 (6502.82
seconds) Cbc0010I After 77000 nodes, 21185 on tree, 872 best solution,
best possible 749.62893 (6505.93 seconds) Cbc0010I After 77100 nodes,
21192 on tree, 872 best solution, best possible 749.6553 (6512.37 seconds)
```

```
Cbc0010I After 77200 nodes, 21191 on tree, 872 best solution, best
possible 749.6553 (6517.76 seconds) Cbc0010I After 77300 nodes, 21191 on
tree, 872 best solution, best possible 749.6553 (6523.31 seconds) Cbc0010I
After 77400 nodes, 21196 on tree, 872 best solution, best possible
749.6553 (6527.61 seconds) Cbc0010I After 77500 nodes, 21199 on tree, 872
best solution, best possible 749.6553 (6530.69 seconds) Cbc0010I After
77600 nodes, 21196 on tree, 872 best solution, best possible 749.6553
(6533.53 seconds) Cbc0010I After 77700 nodes, 21193 on tree, 872 best
solution, best possible 749.6553 (6536.99 seconds) Cbc0010I After 77800
nodes, 21199 on tree, 872 best solution, best possible 749.6553 (6542.33
seconds) Cbc0010I After 77900 nodes, 21197 on tree, 872 best solution,
best possible 749.6553 (6546.42 seconds) Cbc0010I After 78000 nodes, 21191
on tree, 872 best solution, best possible 749.6553 (6550.92 seconds)
Cbc0010I After 78100 nodes, 21228 on tree, 872 best solution, best
possible 749.6553 (6560.61 seconds) Cbc0010I After 78200 nodes, 21271 on
tree, 872 best solution, best possible 749.6553 (6571.76 seconds) Cbc0010I
After 78300 nodes, 21314 on tree, 872 best solution, best possible
749.6553 (6582.09 seconds) Cbc0010I After 78400 nodes, 21356 on tree, 872
best solution, best possible 749.6553 (6592.72 seconds) Cbc0010I After
78500 nodes, 21397 on tree, 872 best solution, best possible 749.6553
(6601.92 seconds) Cbc0010I After 78600 nodes, 21439 on tree, 872 best
solution, best possible 749.6553 (6611.17 seconds) Cbc0010I After 78700
nodes, 21481 on tree, 872 best solution, best possible 749.6553 (6619.45
seconds) Cbc0010I After 78800 nodes, 21525 on tree, 872 best solution,
best possible 749.6553 (6627.80 seconds) Cbc0010I After 78900 nodes, 21567
on tree, 872 best solution, best possible 749.6553 (6638.92 seconds)
Cbc0010I After 79000 nodes, 21605 on tree, 872 best solution, best
possible 749.6553 (6648.49 seconds) Cbc0010I After 79100 nodes, 21646 on
tree, 872 best solution, best possible 749.6553 (6657.51 seconds) Cbc0010I
After 79200 nodes, 21689 on tree, 872 best solution, best possible
749.6553 (6666.44 seconds) Cbc0010I After 79300 nodes, 21734 on tree, 872
best solution, best possible 749.6553 (6677.15 seconds) Cbc0010I After
79400 nodes, 21776 on tree, 872 best solution, best possible 749.6553
(6687.32 seconds) Cbc0010I After 79500 nodes, 21817 on tree, 872 best
solution, best possible 749.6553 (6697.31 seconds) Cbc0010I After 79600
nodes, 21858 on tree, 872 best solution, best possible 749.6553 (6707.95
seconds) Cbc0010I After 79700 nodes, 21902 on tree, 872 best solution,
best possible 749.6553 (6718.54 seconds) Cbc0010I After 79800 nodes, 21946
on tree, 872 best solution, best possible 749.6553 (6729.18 seconds)
Cbc0010I After 79900 nodes, 21992 on tree, 872 best solution, best
possible 749.6553 (6738.41 seconds) Cbc0010I After 80000 nodes, 22038 on
tree, 872 best solution, best possible 749.6553 (6748.60 seconds) Cbc0010I
After 80100 nodes, 22040 on tree, 872 best solution, best possible
749.6553 (6751.44 seconds) Cbc0010I After 80200 nodes, 22032 on tree, 872
best solution, best possible 749.6553 (6754.74 seconds) Cbc0010I After
80300 nodes, 22025 on tree, 872 best solution, best possible 749.6553
(6757.98 seconds) Cbc0010I After 80400 nodes, 22026 on tree, 872 best
solution, best possible 749.6553 (6761.72 seconds) Cbc0010I After 80500
nodes, 22018 on tree, 872 best solution, best possible 749.6553 (6764.30
seconds) Cbc0010I After 80600 nodes, 22009 on tree, 872 best solution,
best possible 749.6553 (6767.49 seconds) Cbc0010I After 80700 nodes, 22002
on tree, 872 best solution, best possible 749.6553 (6770.73 seconds)
Cbc0010I After 80800 nodes, 22007 on tree, 872 best solution, best
possible 749.6553 (6773.82 seconds) Cbc0010I After 80900 nodes, 21998 on
tree, 872 best solution, best possible 749.6553 (6777.48 seconds) Cbc0010I
After 81000 nodes, 22005 on tree, 872 best solution, best possible
749.6553 (6780.85 seconds) Cbc0010I After 81100 nodes, 22022 on tree, 872
best solution, best possible 749.6553 (6786.00 seconds) Cbc0010I After
81200 nodes, 22018 on tree, 872 best solution, best possible 749.6553
(6789.73 seconds) Cbc0010I After 81300 nodes, 22020 on tree, 872 best
solution, best possible 749.6553 (6794.00 seconds) Cbc0010I After 81400
nodes, 22028 on tree, 872 best solution, best possible 749.6553 (6798.75
seconds) Cbc0010I After 81500 nodes, 22022 on tree, 872 best solution,
best possible 749.6553 (6803.30 seconds) Cbc0010I After 81600 nodes, 22023
on tree, 872 best solution, best possible 749.6553 (6807.11 seconds)
Cbc0010I After 81700 nodes, 22025 on tree, 872 best solution, best
possible 749.6553 (6809.88 seconds) Cbc0010I After 81800 nodes, 22026 on
tree, 872 best solution, best possible 749.6553 (6813.29 seconds) Cbc0010I
After 81900 nodes, 22019 on tree, 872 best solution, best possible
```

```
    749.6553 (6817.96 seconds) Cbc0010I After 82000 nodes, 22021 on tree, 872
    best solution, best possible 749.6553 (6821.38 seconds) Cbc0010I After
    82100 nodes, 22062 on tree, 872 best solution, best possible 749.6553
    (6833.30 seconds) Cbc0010I After 82200 nodes, 22103 on tree, 872 best
    solution, best possible 749.6553 (6843.10 seconds) Cbc0010I After 82300
    nodes, 22149 on tree, 872 best solution, best possible 749.6553 (6852.36
    seconds) Cbc0010I After 82400 nodes, 22193 on tree, 872 best solution,
    best possible 749.6553 (6863.33 seconds) Cbc0010I After 82500 nodes, 22238
    on tree, 872 best solution, best possible 749.6553 (6873.51 seconds)
    Cbc0010I After 82600 nodes, 22281 on tree, 872 best solution, best
    possible 749.6553 (6882.52 seconds) Cbc0010I After 82700 nodes, 22322 on
    tree, 872 best solution, best possible 749.6553 (6893.36 seconds) Cbc0010I
    After 82800 nodes, 22358 on tree, 872 best solution, best possible
    749.6553 (6903.55 seconds) Cbc0010I After 82900 nodes, 22405 on tree, 872
    best solution, best possible 749.6553 (6913.78 seconds) Cbc0010I After
    83000 nodes, 22451 on tree, 872 best solution, best possible 749.6553
    (6924.79 seconds) Cbc0010I After 83100 nodes, 22490 on tree, 872 best
    solution, best possible 749.6553 (6935.14 seconds) Cbc0010I After 83200
    nodes, 22530 on tree, 872 best solution, best possible 749.6553 (6945.03
    seconds) Cbc0010I After 83300 nodes, 22574 on tree, 872 best solution,
    best possible 749.6553 (6954.64 seconds) Cbc0010I After 83400 nodes, 22614
    on tree, 872 best solution, best possible 749.6553 (6963.72 seconds)
    Cbc0010I After 83500 nodes, 22656 on tree, 872 best solution, best
    possible 749.6553 (6972.70 seconds) Cbc0010I After 83600 nodes, 22701 on
    tree, 872 best solution, best possible 749.6553 (6983.43 seconds) Cbc0010I
    After 83700 nodes, 22747 on tree, 872 best solution, best possible
    749.6553 (6992.03 seconds) Cbc0010I After 83800 nodes, 22787 on tree, 872
    best solution, best possible 749.6553 (7002.54 seconds) Cbc0010I After
    83900 nodes, 22834 on tree, 872 best solution, best possible 749.6553
    (7014.11 seconds) Cbc0010I After 84000 nodes, 22879 on tree, 872 best
    solution, best possible 749.6553 (7024.53 seconds) Cbc0010I After 84100
    nodes, 22880 on tree, 872 best solution, best possible 749.6553 (7027.31
    seconds) Cbc0010I After 84200 nodes, 22875 on tree, 872 best solution,
    best possible 749.6553 (7030.38 seconds) Cbc0010I After 84300 nodes, 22869
    on tree, 872 best solution, best possible 749.6553 (7032.94 seconds)
    Cbc0010I After 84400 nodes, 22867 on tree, 872 best solution, best
    possible 749.6553 (7036.96 seconds)

---------------------------------------------------------------------------
ApplicationError                          Traceback (most recent call last)
<ipython-input-30-023d2fee3b0f> in <module>()
     30     job += 1
     31
---> 32 Visualize(JobShop(TASKS))

<ipython-input-29-8a3da2f5248d> in JobShop(TASKS, tclean, ZW)
     33
     34     TransformationFactory('gdp.chull').apply_to(model)
---> 35     solver.solve(model)
     36
     37     results = [{'Job': j,

/usr/local/lib/python3.6/dist-packages/pyomo/opt/base/solvers.py in solve(self, *args,
**kwds)
    624                     logger.error("Solver log:\n" + str(_status.log))
    625                 raise pyutilib.common.ApplicationError(
--> 626                     "Solver (%s) did not exit normally" % self.name)
    627             solve_completion_time = time.time()
    628             if self._report_timing:

ApplicationError: Solver (cbc) did not exit normally
```

Recalculate Benchmark Problem with a Zero-Wait Policy

The following calculation is quite intensive and will take several minutes to finish with the `gurobi` solver.

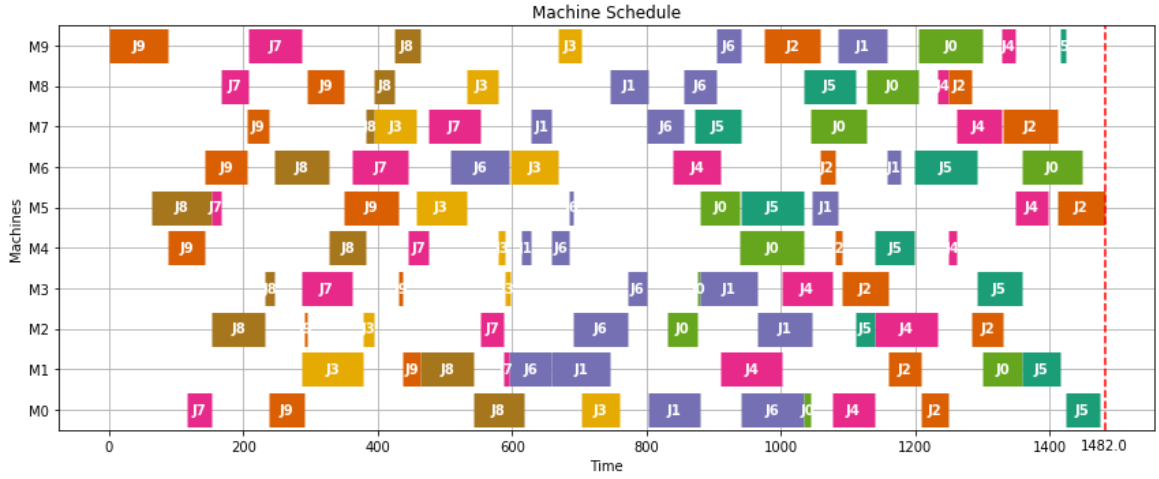
In [ ]:

```
Visualize(JobShop(TASKS, ZW=True))
```

```
Academic license - for non-commercial use only
Optimize a model with 4241 rows, 2802 columns and 10281 nonzeros
Variable types: 1902 continuous, 900 integer (900 binary)
Coefficient statistics:
  Matrix range     [1e+00, 5e+03]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 5e+03]
  RHS range        [1e+00, 1e+02]
Presolve removed 1531 rows and 1441 columns
Presolve time: 0.02s
Presolved: 2710 rows, 1361 columns, 8120 nonzeros
Variable types: 911 continuous, 450 integer (450 binary)

Root relaxation: objective 6.200000e+02, 485 iterations, 0.01 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  620.00000    0   64          -  620.00000      -     -    0s
     0     0  640.56756    0   87          -  640.56756      -     -    0s
     0     0  650.03766    0   94          -  650.03766      -     -    0s
     0     0  650.03766    0   88          -  650.03766      -     -    0s
     0     0  669.53768    0  108          -  669.53768      -     -    0s
     0     0  670.11736    0   91          -  670.11736      -     -    0s
     0     0  670.11736    0   92          -  670.11736      -     -    0s
     0     0  671.34649    0   82          -  671.34649      -     -    0s
     0     0  671.34649    0  110          -  671.34649      -     -    0s
     0     0  671.55468    0  101          -  671.55468      -     -    0s
     0     0  671.60312    0   97          -  671.60312      -     -    0s
     0     0  671.60312    0   98          -  671.60312      -     -    0s
     0     0  671.60312    0  109          -  671.60312      -     -    0s
     0     0  671.60312    0   91          -  671.60312      -     -    0s
     0     0  671.60312    0  102          -  671.60312      -     -    0s
     0     0  671.60312    0   81          -  671.60312      -     -    0s
     0     0  671.60312    0   99          -  671.60312      -     -    0s
     0     0  671.60312    0   99          -  671.60312      -     -    0s
     0     2  671.60312    0   91          -  671.60312      -     -    1s
*  980   641             243    2284.0000000  707.00000  69.0%  38.0    1s
  1092   696 1185.00000  151   64 2284.00000  707.00000  69.0%  37.5    5s
  1117   713  917.00000   76   97 2284.00000  714.59422  68.7%  36.7   10s
  1162   743  904.00000   16  105 2284.00000  837.00000  63.4%  53.5   15s
* 1271   758              29    2258.0000000  837.00000  62.9%  59.3   17s
* 2014   729              80    2174.0000000  837.00000  61.5%  47.7   18s
* 2017   696              80    2155.0000000  837.00000  61.2%  47.7   18s
* 2254   704              77    2131.0000000  837.00000  60.7%  48.2   18s
* 2992   897              78    2099.0000000  837.00000  60.1%  45.0   19s
  3691  1280 1229.00000   24  112 2099.00000  837.00000  60.1%  42.9   20s
* 5173  2022              49    1952.0000000  837.00000  57.1%  41.8   21s
* 6455  2683              56    1945.0000000  915.00000  53.0%  41.1   22s
  8670  4050 1248.00000   22  104 1945.00000  951.00000  51.1%  40.6   25s
* 9764  4687              65    1940.0000000  966.00000  50.2%  42.0   26s
* 9770  4668              69    1931.0000000  966.00000  50.0%  42.0   26s
*10409  5092              53    1922.0000000  966.00000  49.7%  41.5   27s
*10596  5096              68    1897.0000000  966.00000  49.1%  41.4   27s
*12115  6015              84    1874.0000000  966.00000  48.5%  41.5   29s
 13356  6794 1048.00000   19   99 1874.00000  985.00000  47.4%  40.8   30s
 13381  6807 1212.00000   22  102 1874.00000  985.00000  47.4%  41.0   35s
*13394  6679              43    1829.0000000  985.00000  46.1%  41.0   36s
*13433  6564              46    1799.0000000  985.00000  45.2%  41.2   36s
*13484  6572              28    1798.0000000  985.00000  45.2%  41.5   37s
*13518  6098              34    1724.0000000  985.00000  42.9%  41.7   38s
*13573  5468              42    1643.0000000 1002.00000  39.0%  42.0   38s
 13651  5470 1389.00000   29   46 1643.00000 1007.00000  38.7%  42.9   40s
 16123  6399 1332.00000   26   61 1643.00000 1031.00000  37.2%  45.5   46s
```

```
 17506  6866 1580.00000   28   78 1643.00000 1046.00000  36.3%  47.1   50s
 21928  8374 infeasible   36      1643.00000 1084.00000  34.0%  50.4   55s
*23405  8655               44    1641.0000000 1094.12976  33.3%  51.8   57s
 25236  9220 1368.00000   23   79 1641.00000 1106.00000  32.6%  53.0   60s
*28702  9838               27    1621.0000000 1134.00000  30.0%  54.4   64s
 28752  9768 1435.00000   30   38 1621.00000 1134.00000  30.0%  54.8   65s
 29759 10079 1311.00000   61   99 1621.00000 1134.00000  30.0%  55.0   81s
 29764 10082 1564.00000   50  143 1621.00000 1134.00000  30.0%  55.0   85s
 29773 10088 1605.00000   50  157 1621.00000 1134.00000  30.0%  54.9   90s
 29781 10094 1189.23008   22  170 1621.00000 1134.00000  30.0%  54.9   95s
 29789 10099 1489.00000   48  179 1621.00000 1134.00000  30.0%  54.9  100s
 29799 10106 1526.00000   38  204 1621.00000 1134.00000  30.0%  54.9  105s
 29804 10109 1219.00000   27  140 1621.00000 1134.00000  30.0%  54.9  110s
 29812 10114 1588.00000   42  131 1621.00000 1134.00000  30.0%  54.9  115s
 29819 10119 1493.00000   43  169 1621.00000 1134.00000  30.0%  54.9  120s
 29825 10123 1265.00000   39  184 1621.00000 1134.00000  30.0%  54.8  125s
 29831 10127 1255.00000   30  181 1621.00000 1134.00000  30.0%  54.8  130s
 29839 10137 1134.00000   26  144 1621.00000 1134.00000  30.0%  57.0  135s
 29946 10170 1134.00000   34  124 1621.00000 1134.00000  30.0%  57.4  142s
 30048 10196 1134.00000   39  113 1621.00000 1134.00000  30.0%  57.6  145s
 30266 10206 1163.00000   52   74 1621.00000 1134.00000  30.0%  57.9  155s
 34550 10713 1134.00000   47   87 1621.00000 1134.00000  30.0%  55.7  160s
 38097 10754 1219.00000   38  107 1621.00000 1134.00000  30.0%  54.6  165s
*41661  9683               69    1497.0000000 1134.00000  24.2%  53.9  168s
 42973  9420 1251.00000   45  126 1497.00000 1134.00000  24.2%  53.9  170s
 46829  8371 infeasible   44      1497.00000 1163.00000  22.3%  54.1  175s
 50556  6656 infeasible   52      1497.00000 1237.00000  17.4%  54.8  180s
*52349  4939               40    1491.0000000 1280.00000  14.2%  55.1  182s
 53561  3617 infeasible   43      1491.00000 1325.00000  11.1%  55.6  185s
*53924  2654               32    1482.0000000 1343.00000  9.38%  55.8  185s

Cutting planes:
  Gomory: 26
  Cover: 3
  Implied bound: 144
  Projected implied bound: 4
  Clique: 1
  MIR: 12
  StrongCG: 1
  Flow cover: 111
  Inf proof: 111

Explored 56061 nodes (3129028 simplex iterations) in 188.12 seconds
Thread count was 8 (of 8 available processors)

Solution count 10: 1482 1491 1497 ... 1829

Optimal solution found (tolerance 1.00e-04)
Best objective 1.482000000000e+03, best bound 1.482000000000e+03, gap 0.0000%
Freed default Gurobi environment
```


Job Schedule
Jobs
J9
J8
J7
J6
J5
J4
J3
J2
J1
J0
0
200
400
600
800
1000
1200
1400
1482.0
Time


Machine Schedule
Machines
M9
M8
M7
M6
M5
M4
M3
M2
M1
M0
0
200
400
600
800
1000
1200
1400
1482.0
Time

In [ ]:

4.3 Maintenance Planning

Problem Statement

A process unit is operating over a maintenance planning horizon from 1 to T days. On day t the unit makes a profit $c[t]$ which is known in advance. The unit needs to shut down for P maintenance periods during the planning period. Once started, a maintenance period takes M days to finish.

Find a maintenance schedule that allows the maximum profit to be produced.

Modeling with Disjunctive Constraints

The model is comprised of two sets of the binary variables indexed 1 to T. Binary variables x_t correspond to the operating mode of the process unit, with $x_t = 1$ indicating the unit is operating on day t and able to earn a profit c_t. Binary variable $y_t = 1$ indicates the first day of a maintenance period during which the unit is not operating and earning 0 profit.

Objective

The planning objective is to maximize profit realized during the days the plant is operational.

$$\text{Profit} = \max_{x,y} \sum_{t=1}^{T} c_t x_t$$

subject to completing P maintenance periods.

Constraints

Number of planning periods is equal to P.

Completing P maintenance periods requires a total of P starts.

$$\sum_{t=1}^{T} y_t = P$$

No more than one maintenance period can start in any consecutive set of M days.

No more than one maintenance period can start in any consecutive set of M days.

$$\sum_{s=0}^{M-1} y_{t+s} \leq 1 \qquad \forall t = 1, 2, \ldots, T - M + 1$$

This last requirement could be modified if some period of time should occur between maintenance periods.

The unit must shut down for M days following a maintenance start.

The final requirement is a disjunctive constraint that says either $y_t = 0$ or the sum $\sum_{s}^{M-1} x_{t+s} = 0$, but not both. Mathematically, this forms a set of constraints reading

$$\left(y_t = 0\right) \vee \left(\sum_{s=0}^{M-1} x_{t+s} = 0\right) \qquad \forall t = 1, 2, \ldots, T - M + 1$$

where $\vee$ denotes a disjunction.

Disjunctive constraints of this nature are frequently encountered in scheduling problems. In this particular case, the disjunctive constraints can be replaced by a set of linear inequalities using the Big-M method.

$$\sum_{s=0}^{M-1} x_{t+s} \leq M(1 - y_t) \qquad \forall t = 1, 2, \ldots, T - M + 1$$

In this case, the M appearing on the right-hand side of this constraint happens to be the same as the length of each maintenance period.

Pyomo Solution using the Big-M Method

Initialization

If you are using this notebook in Google Colaboratory, the following cell will install Pyomo and the COIN-OR CBC solver needed to execute the code in this notebook. Otherwise you should verify that Pyomo and the COIN-OR CBC solver have been successfully installed before attempting to run the code in this notebook.

In [1]:

```
import sys
if 'google.colab' in sys.modules:
    !pip install -q pyomo
    !apt-get install -y -qq coinor-cbc
```

Select Solver

The next cell constructs the SolverFactory() object that will be used in subsequent calculations in this notebook. Doing this at the start makes it simpler to adapt this notebook to different solvers and computational environments.

In [2]:

```
import pyomo.environ as pyo
solver = pyo.SolverFactory('cbc')
```

Parameter Values

In [3]:

```
import numpy as np

# problem parameters
T = 90          # planning period from 1..T
M = 3           # length of maintenance period
P = 4           # number of maintenance periods

# daily profits
c = {k:np.random.uniform() for k in range(1, T+1)}
```

Pyomo Model

The disjunctive constraints can be represented directly in Pyomo using the Generalized Disjunctive Programming extension. The GDP extension transforms the disjunctive constraints to an MILP using convex hull and cutting plane methods.

In [4]:

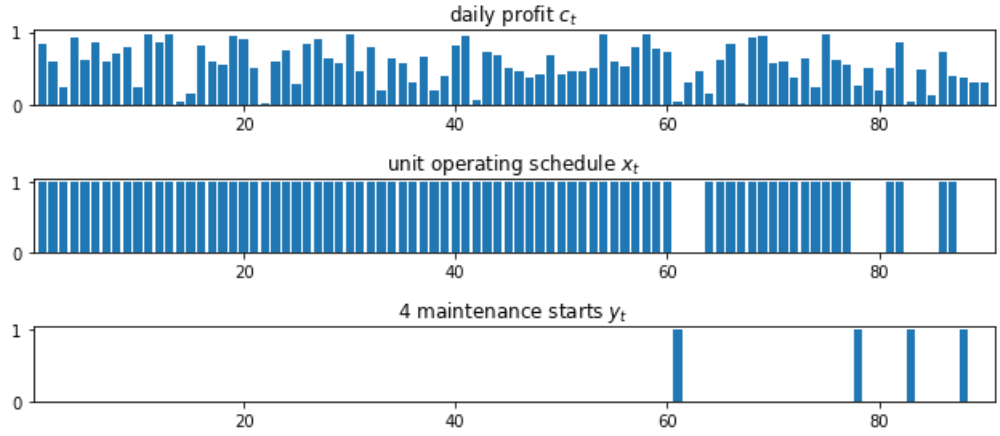
```
import pyomo.environ as pyo
import matplotlib.pyplot as plt

def maintenance_planning_bigm(c, T, M, P):
    m = pyo.ConcreteModel()

    m.T = pyo.RangeSet(1, T)
    m.Y = pyo.RangeSet(1, T - M + 1)
    m.S = pyo.RangeSet(0, M - 1)

    m.c = pyo.Param(m.T, initialize = c)
    m.x = pyo.Var(m.T, domain=pyo.Binary)
    m.y = pyo.Var(m.T, domain=pyo.Binary)

    # objective
    m.profit = pyo.Objective(expr = sum(m.c[t]*m.x[t] for t in m.T), sense=pyo.maximize)

    # required number P of maintenance starts
    m.sumy = pyo.Constraint(expr = sum(m.y[t] for t in m.Y) == P)

    # no more than one maintenance start in the period of length M
    m.sprd = pyo.Constraint(m.Y, rule = lambda m, t: sum(m.y[t+s] for s in m.S) <= 1)

    # disjunctive constraints
    m.bigm = pyo.Constraint(m.Y, rule = lambda m, t: sum(m.x[t+s] for s in m.S) <= M*(1 -
m.y[t]))

    return m

m = maintenance_planning_bigm(c, T, M, P)
pyo.SolverFactory('glpk').solve(m).write()
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: 47.0700514715296
  Upper bound: 47.0700514715296
  Number of objectives: 1
  Number of constraints: 178
  Number of variables: 181
  Number of nonzeros: 705
  Sense: maximize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Termination condition: optimal
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 13021
      Number of created subproblems: 13021
  Error rc: 0
  Time: 1.807023048400879
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

Display Results

In [5]:

```
def plot_schedule(m):
    fig,ax = plt.subplots(3,1, figsize=(9,4))

    ax[0].bar(m.T, [m.c[t] for t in m.T])
    ax[0].set_title('daily profit $c_t$')

    ax[1].bar(m.T, [m.x[t]() for t in m.T], label='normal operation')
    ax[1].set_title('unit operating schedule $x_t$')

    ax[2].bar(m.Y, [m.y[t]() for t in m.Y])
    ax[2].set_title(str(P) + ' maintenance starts $y_t$')
    for a in ax:
        a.set_xlim(0.1, len(m.T)+0.9)

    plt.tight_layout()

plot_schedule(m)
```



Pyomo Solution using the Generalized Disjunctive Constraints Extension

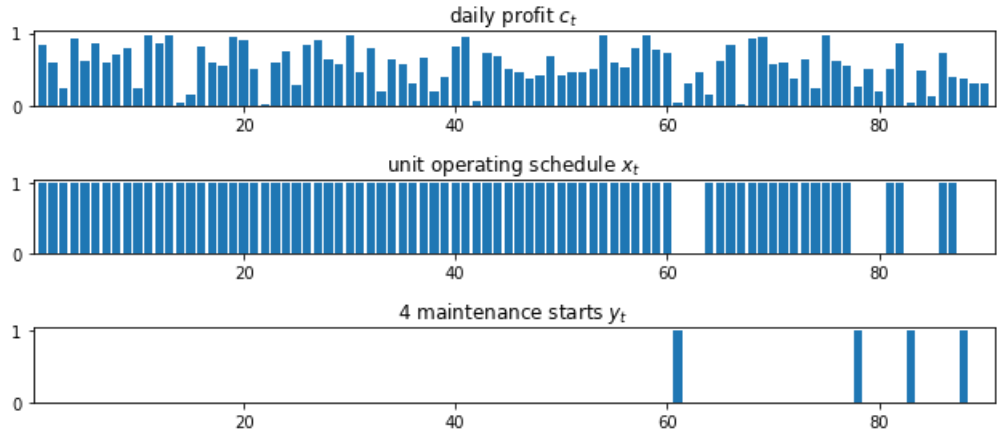
Disjunctive constraints can be represented directly in Pyomo using the Generalized Disjunctive Programming extension. The advantage of using the extension is that constraints can be transformed to an MILP using alternatives to the big-M, such as convex hull and cutting plane methods.

The following cell replaces the Big-M constraints with disjunctions. Disjunctions are represented by lists of mutually exclusive constraints.

In [6]:

```
import pyomo.environ as pyo
import pyomo.gdp as gdp
import matplotlib.pyplot as plt

def maintenance_planning_gdp(c, T, M, P):
    m = pyo.ConcreteModel()

    m.T = pyo.RangeSet(1, T)
    m.Y = pyo.RangeSet(1, T - M + 1)
    m.S = pyo.RangeSet(0, M - 1)

    m.c = pyo.Param(m.T, initialize = c)
    m.x = pyo.Var(m.T, domain=pyo.Binary)
    m.y = pyo.Var(m.T, domain=pyo.Binary)

    # objective
    m.profit = pyo.Objective(expr = sum(m.c[t]*m.x[t] for t in m.T), sense=pyo.maximize)

    # required number P of maintenance starts
    m.sumy = pyo.Constraint(expr = sum(m.y[t] for t in m.Y) == P)

    # no more than one maintenance start in the period of length M
    m.sprd = pyo.Constraint(m.Y, rule = lambda m, t: sum(m.y[t+s] for s in m.S) <= 1)

    # disjunctive constraints
    m.disj = gdp.Disjunction(m.Y, rule = lambda m, t: [m.y[t]==0, sum(m.x[t+s] for s in m
.S)==0])

    # transformation and soluton
    pyo.TransformationFactory('gdp.chull').apply_to(m)

    return m

m = maintenance_planning_gdp(c, T, M, P)
solver.solve(m).write()
plot_schedule(m)
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: -47.07005147
  Upper bound: -47.07005147
  Number of objectives: 1
  Number of constraints: 440
  Number of variables: 266
  Number of binary variables: 356
  Number of integer variables: 356
  Number of nonzeros: 90
  Sense: maximize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  User time: -1.0
  System time: 0.1
  Wallclock time: 0.11
  Termination condition: optimal
  Termination message: Model was solved to optimality (subject to tolerances), and an
optimal solution is available.
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 0
      Number of created subproblems: 0
    Black box:
      Number of iterations: 0
  Error rc: 0
  Time: 0.12743568420410156
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```



Ramping Constraints

Prior to maintenance shutdown, a large processing unit may take some time to safely ramp down from full production. And then require more time to safely ramp back up to full production following maintenace. To provide for ramp-down and ramp-up periods, we modify the problem formation in the following ways.

- The variable denoting unit operation, x_t is changed from a binary variable to a continuous variable $\0 \le x_t \le 1$ denoting the fraction of total capacity at which the unit is operating on day t.
- Two new variable sequences, $0 \le u_t^+ \le u_t^{+,\max}$ and $0 \le u_t^- \le u_t^{-,\max}$, are introduced which denote the fraction increase or decrease in unit capacity to completed on day t.
- An additional sequence of equality constraints is introduced relating x_t to u_t^+ and u_t^-.

$$x_t = x_{t-1} + u_t^+ - u_t^-$$

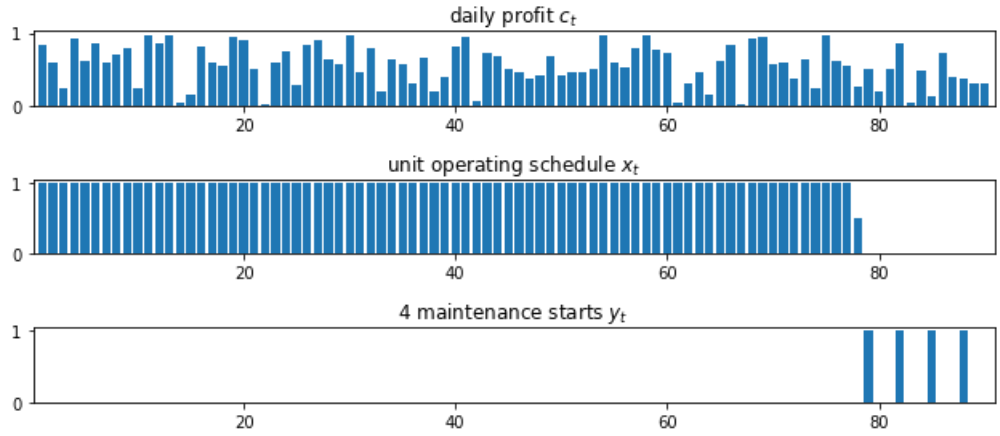
We begin the Pyomo model by specifying the constraints, then modifying the Big-M formulation to add the features described above.

In [7]:

```
upos_max = 0.3334
uneg_max = 0.5000
```

In [8]:

```
import pyomo.environ as pyo
import pyomo.gdp as gdp
import matplotlib.pyplot as plt

def maintenance_planning_ramp(c, T, M, P):
    m = pyo.ConcreteModel()

    m.T = pyo.RangeSet(1, T)
    m.Y = pyo.RangeSet(1, T - M + 1)
    m.S = pyo.RangeSet(0, M - 1)

    m.c = pyo.Param(m.T, initialize = c)
    m.x = pyo.Var(m.T, bounds=(0, 1))
    m.y = pyo.Var(m.T, domain=pyo.Binary)
    m.upos = pyo.Var(m.T, bounds=(0, upos_max))
    m.uneg = pyo.Var(m.T, bounds=(0, uneg_max))

    # objective
    m.profit = pyo.Objective(expr = sum(m.c[t]*m.x[t] for t in m.T), sense=pyo.maximize)

    # ramp constraint
    m.ramp = pyo.Constraint(m.T, rule = lambda m, t:
         m.x[t] == m.x[t-1] + m.upos[t] - m.uneg[t] if t > 1 else pyo.Constraint.Skip)

    # required number P of maintenance starts
    m.sumy = pyo.Constraint(expr = sum(m.y[t] for t in m.Y) == P)

    # no more than one maintenance start in the period of length M
    m.sprd = pyo.Constraint(m.Y, rule = lambda m, t: sum(m.y[t+s] for s in m.S) <= 1)

    # disjunctive constraints
    m.disj = gdp.Disjunction(m.Y, rule = lambda m, t: [m.y[t]==0, sum(m.x[t+s] for s in m
.S)==0])

    # transformation and soluton
    pyo.TransformationFactory('gdp.chull').apply_to(m)

    return m

m = maintenance_planning_ramp(c, T, M, P)
solver.solve(m)
plot_schedule(m)
```



Introducing a Minimum Number of Operational Days between Maintenance Periods

Up to this point we have imposed no constraints on the frequency of maintenance periods. Without such constraints, particularly when ramping constraints are imposed, is that maintenance periods will be scheduled back-to-back, which is clearly not a useful result for most situations.

The next revision of the model is to incorporate a requirement that N operational days be scheduled between any mainteance periods. This does allow for maintenance to be postponed until the very end of the planning period. The disjunctive constraints read

$$\left(y_t = 0\right) \vee \left(\sum_{s=0}^{(M+N-1) \wedge (t+s \le T)} x_{t+s} = 0 \right) \qquad \forall t = 1, 2, \ldots, T - M + 1$$

where the upper bound on the summation is needed to handle the terminal condition.

Paradoxically, this is an example where the Big-M method provides a much faster solution.

$$\sum_{s=0}^{(M+N-1) \wedge (t+s \le T)} x_{t+s} \le (M+N)(1 - y_t) \qquad \forall t = 1, 2, \ldots, T - M + 1$$

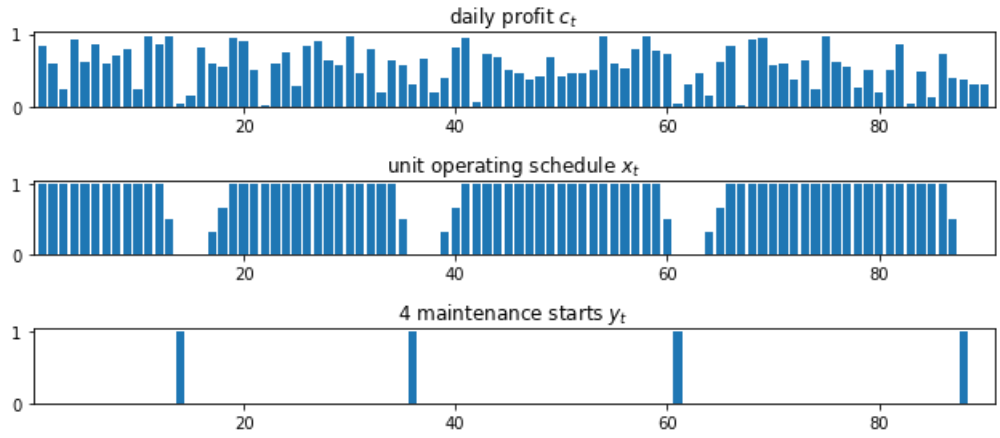
The following cell implements both sets of constraints.

In [9]:

```
N = 10  # minimum number of operational days between maintenance periods
```

In [10]:

```
import pyomo.environ as pyo
import pyomo.gdp as gdp
import matplotlib.pyplot as plt

def maintenance_planning_ramp_operational(c, T, M, P, N):
    m = pyo.ConcreteModel()

    m.T = pyo.RangeSet(1, T)
    m.Y = pyo.RangeSet(1, T - M + 1)
    m.S = pyo.RangeSet(0, M - 1)
    m.W = pyo.RangeSet(0, M + N - 1)

    m.c = pyo.Param(m.T, initialize = c)
    m.x = pyo.Var(m.T, bounds=(0, 1))
    m.y = pyo.Var(m.T, domain=pyo.Binary)
    m.upos = pyo.Var(m.T, bounds=(0, upos_max))
    m.uneg = pyo.Var(m.T, bounds=(0, uneg_max))

    # objective
    m.profit = pyo.Objective(expr = sum(m.c[t]*m.x[t] for t in m.T), sense=pyo.maximize)

    # ramp constraint
    m.ramp = pyo.Constraint(m.T, rule = lambda m, t:
         m.x[t] == m.x[t-1] + m.upos[t] - m.uneg[t] if t > 1 else pyo.Constraint.Skip)

    # required number P of maintenance starts
    m.sumy = pyo.Constraint(expr = sum(m.y[t] for t in m.Y) == P)

    # no more than one maintenance start in the period of length M
    m.sprd = pyo.Constraint(m.Y, rule = lambda m, t: sum(m.y[t+s] for s in m.W if t + s <
= T) <= 1)

    # Choose one or the other the following methods. Comment out the method not used.

    # disjunctive constraints, big-M method.
    m.bigm = pyo.Constraint(m.Y, rule = lambda m, t: sum(m.x[t+s] for s in m.S) <= (M+N)*
(1 - m.y[t]))

    # disjunctive constraints, GDP programming method
    #m.disj = gdp.Disjunction(m.Y, rule = lambda m, t: [m.y[t]==0, sum(m.x[t+s] for s
in m.W if t + s <= T)==0])
    #pyo.TransformationFactory('gdp.chull').apply_to(m)

    return m

m = maintenance_planning_ramp_operational(c, T, M, P, N)
solver.solve(m)
plot_schedule(m)
```



Exercises

1. Rather than specify how many maintenance periods must be accomodated, modify the model so that the process unit can operate no more than N days without a maintenance shutdown. (Hint. You may to introduce an additional set of binary variables, z_t to denote the start of an operational period.)
2. Do a systematic comparison of the Big-M, Convex Hull, and Cutting Plane techniques for implementing the disjunctive constraints. Your comparison should include a measure of complexity (such as the number of decision variables and constraints in the resulting transformed problems), computational effort, and the effect of solver (such as glpk vs cbc).

4.4 Scheduling Multipurpose Batch Processes using State-Task Networks

References

Floudas, C. A., & Lin, X. (2005). Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. Annals of Operations Research, 139(1), 131-162.

Harjunkoski, I., Maravelias, C. T., Bongers, P., Castro, P. M., Engell, S., Grossmann, I. E., ... & Wassick, J. (2014). Scope for industrial applications of production scheduling models and solution methods. Computers & Chemical Engineering, 62, 161-193.

Kondili, E., Pantelides, C. C., & Sargent, R. W. H. (1993). A general algorithm for short-term scheduling of batch operations—I. MILP formulation. Computers & Chemical Engineering, 17(2), 211-227.

Méndez, C. A., Cerdá, J., Grossmann, I. E., Harjunkoski, I., & Fahl, M. (2006). State-of-the-art review of optimization methods for short-term scheduling of batch processes. Computers & Chemical Engineering, 30(6), 913-946.

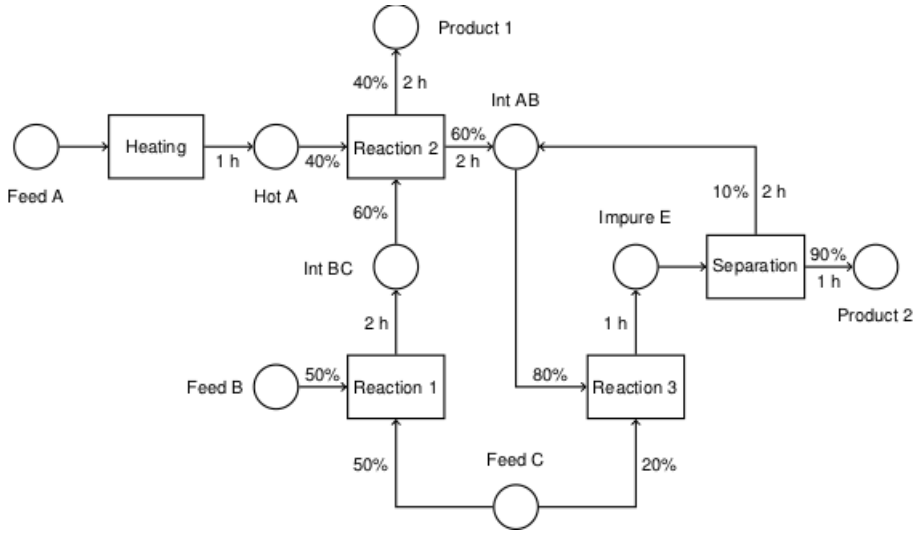
Shah, N., Pantelides, C. C., & Sargent, R. W. H. (1993). A general algorithm for short-term scheduling of batch operations—II. Computational issues. Computers & Chemical Engineering, 17(2), 229-244.

Wassick, J. M., & Ferrio, J. (2011). Extending the resource task network for industrial applications. Computers & chemical engineering, 35(10), 2124-2140.

Example (Kondili, et al., 1993)

A state-task network is a graphical representation of the activities in a multiproduct batch process. The representation includes the minimum details needed for short term scheduling of batch operations.

A well-studied example due to Kondili (1993) is shown below. Other examples are available in the references cited above.



Each circular node in the diagram designates material in a particular state. The materials are generally held in suitable vessels with a known capacity. The relevant information for each state is the initial inventory, storage capacity, and the unit price of the material in each state. The price of materials in intermediate states may be assigned penalities in order to minimize the amount of work in progress.

The rectangular nodes denote process tasks. When scheduled for execution, each task is assigned an appropriate piece of equipment, and assigned a batch of material according to the incoming arcs. Each incoming arc begins at a state where the associated label indicates the mass fraction of the batch coming from that particular state. Outgoing arcs indicate the disposition of the batch to product states. The outgoing are labels indicate the fraction of the batch assigned to each product state, and the time necessary to produce that product.

Not shown in the diagram is the process equipment used to execute the tasks. A separate list of process units is available, each characterized by a capacity and list of tasks which can be performed in that unit.

Exercise

Read this reciped for Hollandaise Sauce: http://www.foodnetwork.com/recipes/tyler-florence/hollandaise-sauce-recipe-1910043. Assume the available equipment consists of one sauce pan and a double-boiler on a stove. Draw a state-task network outlining the basic steps in the recipe.

Encoding the STN data

The basic data structure specifies the states, tasks, and units comprising a state-task network. The intention is for all relevant problem data to be contained in a single JSON-like structure.

In [1]:

```
H = 20

Kondili = {
    'TIME':  range(0,H+1),
    'STATES': {
        'Feed_A'    : {'capacity': 500, 'initial': 500, 'price':  0},
        'Feed_B'    : {'capacity': 500, 'initial': 500, 'price':  0},
        'Feed_C'    : {'capacity': 500, 'initial': 500, 'price':  0},
        'Hot_A'     : {'capacity': 100, 'initial':   0, 'price': -1},
        'Int_AB'    : {'capacity': 200, 'initial':   0, 'price': -1},
        'Int_BC'    : {'capacity': 150, 'initial':   0, 'price': -1},
        'Impure_E' : {'capacity': 100, 'initial':   0, 'price': -1},
        'Product_1': {'capacity': 500, 'initial':   0, 'price': 10},
        'Product_2': {'capacity': 500, 'initial':   0, 'price': 10},
    },
    'ST_ARCS': {
        ('Feed_A',   'Heating')   : {'rho': 1.0},
        ('Feed_B',   'Reaction_1'): {'rho': 0.5},
        ('Feed_C',   'Reaction_1'): {'rho': 0.5},
        ('Feed_C',   'Reaction_3'): {'rho': 0.2},
        ('Hot_A',    'Reaction_2'): {'rho': 0.4},
        ('Int_AB',   'Reaction_3'): {'rho': 0.8},
        ('Int_BC',   'Reaction_2'): {'rho': 0.6},
        ('Impure_E', 'Separation'): {'rho': 1.0},
    },
    'TS_ARCS': {
        ('Heating',    'Hot_A')    : {'dur': 1, 'rho': 1.0},
        ('Reaction_2', 'Product_1'): {'dur': 1.5, 'rho': 0.4},
        ('Reaction_2', 'Int_AB')   : {'dur': 1.5, 'rho': 0.6},
        ('Reaction_1', 'Int_BC')   : {'dur': 2, 'rho': 1.0},
        ('Reaction_3', 'Impure_E') : {'dur': 1, 'rho': 1.0},
        ('Separation', 'Int_AB')   : {'dur': 2, 'rho': 0.1},
        ('Separation', 'Product_2'): {'dur': 1, 'rho': 0.9},
    },
    'UNIT_TASKS': {
        ('Heater',    'Heating')   : {'Bmin': 0, 'Bmax': 100, 'Cost': 1, 'vCost': 0, 'Tc
lean': 0},
        ('Reactor_1', 'Reaction_1'): {'Bmin': 0, 'Bmax':  80, 'Cost': 1, 'vCost': 0, 'Tc
lean': 0},
        ('Reactor_1', 'Reaction_2'): {'Bmin': 0, 'Bmax':  80, 'Cost': 1, 'vCost': 0, 'Tc
lean': 0},
        ('Reactor_1', 'Reaction_3'): {'Bmin': 0, 'Bmax':  80, 'Cost': 1, 'vCost': 0, 'Tc
lean': 0},
        ('Reactor_2', 'Reaction_1'): {'Bmin': 0, 'Bmax':  80, 'Cost': 1, 'vCost': 0, 'Tc
lean': 0},
        ('Reactor_2', 'Reaction_2'): {'Bmin': 0, 'Bmax':  80, 'Cost': 1, 'vCost': 0, 'Tc
lean': 0},
        ('Reactor_2', 'Reaction_3'): {'Bmin': 0, 'Bmax':  80, 'Cost': 1, 'vCost': 0, 'Tc
lean': 0},
        ('Still',     'Separation'): {'Bmin': 0, 'Bmax': 200, 'Cost': 1, 'vCost': 0, 'Tc
lean': 0},
    },
}
```

Setting a Time Grid

The following computations can be done on any time grid, including real-valued time points. TIME is a list of time points commencing at 0.

Creating a Pyomo Model

The following Pyomo model closely follows the development in Kondili, et al. (1993). In particular, the first step in the model is to process the STN data to create sets as given in Kondili.

One important difference from Kondili is the adoption of a more natural time scale that starts at $t = 0$ and extends to $t = H$ (rather than from 1 to H+1).

A second difference is the introduction of an additional decision variable denoted by $Q_{j,t}$ indicating the amount of material in unit j at time t. A material balance then reads

$$Q_{jt} = Q_{j(t-1)} + \sum_{i \in I_j} B_{ijt} - \sum_{i \in I_j} \sum_{\substack{s \in \bar{S}_i \\ s \ni t - P_{is} \geq 0}} \bar{\rho}_{is} B_{ij(t-P_{is})} \qquad \forall j, t$$

Following Kondili's notation, I_j is the set of tasks that can be performed in unit j, and $\bar{S}_i$ is the set of states fed by task j. We assume the units are empty at the beginning and end of production period, i.e.,

$$Q_{j(-1)} = 0 \qquad \forall j$$
$$Q_{j,H} = 0 \qquad \forall j$$

The unit allocation constraints are written the full backward aggregation method described by Shah (1993). The allocation constraint reads

$$\sum_{i \in I_j} \sum_{t'=t}^{t-p_i+1} W_{ijt'} \leq 1 \qquad \forall j, t$$

Each processing unit j is tagged with a minimum and maximum capacity, B_{ij}^{min} and B_{ij}^{max}, respectively, denoting the minimum and maximum batch sizes for each task i. A minimum capacity may be needed to cover heat exchange coils in a reactor or mixing blades in a blender, for example. The capacity may depend on the nature of the task being performed. These constraints are written

$$B_{ij}^{min} W_{ijt} \leq B_{ijt} \leq B_{ij}^{max} W_{ijt} \qquad \forall j, \forall i \in I_j, \forall t$$

Characterization of Tasks

In [2]:

```
STN = Kondili

STATES = STN['STATES']
ST_ARCS = STN['ST_ARCS']
TS_ARCS = STN['TS_ARCS']
UNIT_TASKS = STN['UNIT_TASKS']
TIME = STN['TIME']
H = max(TIME)
```

In [3]:

```
TASKS = set([i for (j,i) in UNIT_TASKS])                                    # set of all tasks

S = {i: set() for i in TASKS}                                               # S[i] input set of st
es which feed task i
for (s,i) in ST_ARCS:
    S[i].add(s)

S_ = {i: set() for i in TASKS}                                              # S_[i] output set of
states fed by task i
for (i,s) in TS_ARCS:
    S_[i].add(s)

rho = {(i,s): ST_ARCS[(s,i)]['rho'] for (s,i) in ST_ARCS}                   # rho[(i,s)] input frac
tion of task i from state s

rho_ = {(i,s): TS_ARCS[(i,s)]['rho'] for (i,s) in TS_ARCS}                  # rho_[(i,s)] output f
raction of task i to state s

P = {(i,s): TS_ARCS[(i,s)]['dur'] for (i,s) in TS_ARCS}                     # P[(i,s)] time for
task i output to state s

p = {i: max([P[(i,s)] for s in S_[i]]) for i in TASKS}                      # p[i] completion time
for task i

K = {i: set() for i in TASKS}                                               # K[i] set of units ca
ble of task i
for (j,i) in UNIT_TASKS:
    K[i].add(j)
```

Characterization of States

In [4]:

```
T = {s: set() for s in STATES}                                              # T[s] set of tasks re
iving material from state s
for (s,i) in ST_ARCS:
    T[s].add(i)

T_ = {s: set() for s in STATES}                                             # set of tasks
producing material for state s
for (i,s) in TS_ARCS:
    T_[s].add(i)

C = {s: STATES[s]['capacity'] for s in STATES}                              # C[s] storage capacity
for state s
```

Characterization of Units

In [5]:

```
UNITS = set([j for (j,i) in UNIT_TASKS])

I = {j: set() for j in UNITS}                                    # I[j] set of tasks p
formed with unit j
for (j,i) in UNIT_TASKS:
    I[j].add(i)

Bmax = {(i,j):UNIT_TASKS[(j,i)]['Bmax'] for (j,i) in UNIT_TASKS}  # Bmax[(i,j)] maximum
capacity of unit j for task i
Bmin = {(i,j):UNIT_TASKS[(j,i)]['Bmin'] for (j,i) in UNIT_TASKS}  # Bmin[(i,j)] minimum
capacity of unit j for task i
```

Pyomo Model

In [6]:

```
from pyomo.environ import *
import numpy as np

TIME = np.array(TIME)

model = ConcreteModel()

model.W = Var(TASKS, UNITS, TIME, domain=Boolean)              # W[i,j,t] 1 if task i
starts in unit j at time t
model.B = Var(TASKS, UNITS, TIME, domain=NonNegativeReals)     # B[i,j,t,] size of batch
assigned to task i in unit j at time t
model.S = Var(STATES.keys(), TIME, domain=NonNegativeReals)    # S[s,t] inventory of sta
te s at time t
model.Q = Var(UNITS, TIME, domain=NonNegativeReals)            # Q[j,t] inventory of
unit j at time t
model.Cost = Var(domain=NonNegativeReals)
model.Value = Var(domain=NonNegativeReals)

# Objective is to maximize the value of the final state (see Kondili, Sec. 5)
model.Obj = Objective(expr = model.Value - model.Cost, sense = maximize)

# Constraints
model.cons = ConstraintList()
model.cons.add(model.Value == sum([STATES[s]['price']*model.S[s,H] for s in STATES]))
model.cons.add(model.Cost == sum([UNIT_TASKS[(j,i)]['Cost']*model.W[i,j,t] +
        UNIT_TASKS[(j,i)]['vCost']*model.B[i,j,t] for i in TASKS for j in K[i] for t in
TIME]))

# unit constraints
for j in UNITS:
    rhs = 0
    for t in TIME:
        # a unit can only be allocated to one task
        lhs = 0
        for i in I[j]:
            for tprime in TIME:
                if tprime >= (t-p[i]+1-UNIT_TASKS[(j,i)]['Tclean']) and tprime <= t:
                    lhs += model.W[i,j,tprime]
        model.cons.add(lhs <= 1)

        # capacity constraints (see Konkili, Sec. 3.1.2)
        for i in I[j]:
            model.cons.add(model.W[i,j,t]*Bmin[i,j] <= model.B[i,j,t])
            model.cons.add(model.B[i,j,t] <= model.W[i,j,t]*Bmax[i,j])

        # unit mass balance
        rhs += sum([model.B[i,j,t] for i in I[j]])
```

```
        for i in I[j]:
            for s in S_[i]:
                if t >= P[(i,s)]:
                    rhs -= rho_[(i,s)]*model.B[i,j,max(TIME[TIME <= t-P[(i,s)]])]
        model.cons.add(model.Q[j,t] == rhs)
        rhs = model.Q[j,t]

    # terminal condition
    model.cons.add(model.Q[j,H] == 0)

# state constraints
for s in STATES.keys():
    rhs = STATES[s]['initial']
    for t in TIME:
        # state capacity constraint
        model.cons.add(model.S[s,t] <= C[s])

        # state mass balanace
        for i in T_[s]:
            for j in K[i]:
                if t >= P[(i,s)]:
                    rhs += rho_[(i,s)]*model.B[i,j,max(TIME[TIME <= t-P[(i,s)]])]

        for i in T[s]:
            rhs -= rho[(i,s)]*sum([model.B[i,j,t] for j in K[i]])
        model.cons.add(model.S[s,t] == rhs)
        rhs = model.S[s,t]

# additional production constraints
model.cons.add(model.S['Product_2',H] >= 250)

SolverFactory('glpk').solve(model).write()
```

```
  Signal handler called from  /Users/jeff/anaconda3/lib/python3.6/subprocess.py
_try_wait 1404
  Waiting...
  Signaled process 30126 with signal 2
ERROR: Solver (glpk) returned non-zero return code (-1)
ERROR: Solver log: GLPSOL: GLPK LP/MIP Solver, v4.65 Parameter(s) specified in
    the command line:
     --write /Users/jeff/Dropbox/Git/ND-Pyomo-
         Cookbook/notebooks/scheduling/tmp0uvmbh6h.glpk.raw --wglp
         /Users/jeff/Dropbox/Git/ND-Pyomo-
         Cookbook/notebooks/scheduling/tmpr8n6to0a.glpk.glp --cpxlp
         /Users/jeff/Dropbox/Git/ND-Pyomo-
         Cookbook/notebooks/scheduling/tmpjuds8idv.pyomo.lp
    Reading problem data from '/Users/jeff/Dropbox/Git/ND-Pyomo-
    Cookbook/notebooks/scheduling/tmpjuds8idv.pyomo.lp'...
    /Users/jeff/Dropbox/Git/ND-Pyomo-
    Cookbook/notebooks/scheduling/tmpjuds8idv.pyomo.lp:5778: warning: lower
    bound of variable 'x1' redefined /Users/jeff/Dropbox/Git/ND-Pyomo-
    Cookbook/notebooks/scheduling/tmpjuds8idv.pyomo.lp:5778: warning: upper
    bound of variable 'x1' redefined 890 rows, 612 columns, 2486 non-zeros 168
    integer variables, all of which are binary 5946 lines were read Writing
    problem data to '/Users/jeff/Dropbox/Git/ND-Pyomo-
    Cookbook/notebooks/scheduling/tmpr8n6to0a.glpk.glp'... 5051 lines were
    written GLPK Integer Optimizer, v4.65 890 rows, 612 columns, 2486 non-
    zeros 168 integer variables, all of which are binary Preprocessing... 459
    rows, 552 columns, 1730 non-zeros 168 integer variables, all of which are
    binary Scaling...
     A: min|aij| =  1.000e-01  max|aij| =  2.000e+02  ratio =  2.000e+03
    GM: min|aij| =  3.162e-01  max|aij| =  3.162e+00  ratio =  1.000e+01 EQ:
    min|aij| =  1.000e-01  max|aij| =  1.000e+00  ratio =  1.000e+01 2N:
    min|aij| =  5.000e-02  max|aij| =  1.600e+00  ratio =  3.200e+01
    Constructing initial basis... Size of triangular part is 459 Solving LP
    relaxation... GLPK Simplex Optimizer, v4.65 459 rows, 552 columns, 1730
    non-zeros
          0: obj =  -0.000000000e+00 inf =   1.600e+02 (2)
        161: obj =   3.783856096e+03 inf =   9.934e-14 (0)
```

```
*    257: obj =   9.121531481e+03 inf =   6.361e-14 (0) 1 OPTIMAL LP
     SOLUTION FOUND Integer optimization begins... Long-step dual simplex
     will be used +   257: mip =     not found yet <=              +inf
     (1; 0) +   818: >>>>>   6.429520833e+03 <=   9.121531481e+03  41.9%
     (67; 0) +  1069: >>>>>   7.386666667e+03 <=   9.121531481e+03  23.5%
     (127; 3) +  1309: >>>>>   8.203750000e+03 <=   9.121531481e+03  11.2%
     (173; 31) +  1592: >>>>>   8.526222222e+03 <=   9.121531481e+03   7.0%
     (191; 100) +  4660: >>>>>   8.635833333e+03 <=   9.121531481e+03
     5.6% (576; 135) + 13019: >>>>>   8.706185185e+03 <=   9.121531481e+03
     4.8% (1608; 313) + 16046: >>>>>   8.725500000e+03 <=   9.121531481e+03
     4.5% (1946; 484) + 33655: >>>>>   8.728851852e+03 <=   9.121531481e+03
     4.5% (4044; 837) + 41870: >>>>>   8.910611111e+03 <=   9.121531481e+03
     2.4% (5037; 978) + 68385: mip =   8.910611111e+03 <=   9.121531481e+03
     2.4% (5793; 4912) + 80975: >>>>>   8.941111111e+03 <=
     9.121531481e+03   2.0% (7227; 5015) + 84651: >>>>>   8.961777778e+03
     <=   9.121531481e+03   1.8% (6533; 7110) +111699: mip =
     8.961777778e+03 <=   9.121531481e+03   1.8% (8670; 8139) +137551: mip
     =   8.961777778e+03 <=   9.121531481e+03   1.8% (11053; 8461) +163408:
     mip =   8.961777778e+03 <=   9.121531481e+03   1.8% (13423; 8769)
     +190751: mip =   8.961777778e+03 <=   9.121531481e+03   1.8% (15740;
     9142) +194516: >>>>>   8.962777778e+03 <=   9.121531481e+03   1.8%
     (16105; 9182) +214583: mip =   8.962777778e+03 <=   9.121531481e+03
     1.8% (17629; 10003) +239892: mip =   8.962777778e+03 <=
     9.121531481e+03   1.8% (19842; 10343) +262539: mip =   8.962777778e+03
     <=   9.121531481e+03   1.8% (21931; 10572) +274398: >>>>>
     8.963777778e+03 <=   9.121531481e+03   1.8% (23019; 10705) Time used:
     60.0 secs.  Memory used: 32.0 Mb. +297658: mip =   8.963777778e+03 <=
     9.121531481e+03   1.8% (24790; 11490) +323501: mip =   8.963777778e+03
     <=   9.121531481e+03   1.8% (26950; 11807) +346369: mip =
     8.963777778e+03 <=   9.121531481e+03   1.8% (29222; 12116) +369526:
     mip =   8.963777778e+03 <=   9.121531481e+03   1.8% (31202; 12437)
     +391286: mip =   8.963777778e+03 <=   9.121531481e+03   1.8% (33135;
     12760) +412573: mip =   8.963777778e+03 <=   9.121531481e+03   1.8%
     (35027; 13074) +433462: mip =   8.963777778e+03 <=   9.121531481e+03
     1.8% (36860; 13360) +451949: mip =   8.963777778e+03 <=
     9.121531481e+03   1.8% (38718; 13624) +471024: mip =   8.963777778e+03
     <=   9.121531481e+03   1.8% (40679; 13856)
```

```
---------------------------------------------------------------------------
ApplicationError                          Traceback (most recent call last)
<ipython-input-6-046334ce38aa> in <module>()
     71 model.cons.add(model.S['Product_2',H] >= 250)
     72
---> 73 SolverFactory('glpk').solve(model).write()

~/anaconda3/lib/python3.6/site-packages/pyomo/opt/base/solvers.py in solve(self, *args,
**kwds)
    624                     logger.error("Solver log:\n" + str(_status.log))
    625                 raise pyutilib.common.ApplicationError(
--> 626                     "Solver (%s) did not exit normally" % self.name)
    627             solve_completion_time = time.time()
    628             if self._report_timing:

ApplicationError: Solver (glpk) did not exit normally
```

Analysis

Profitability

In [ ]:

```
print("Value of State Inventories = {0:12.2f}".format(model.Value()))
print("  Cost of Unit Assignments = {0:12.2f}".format(model.Cost()))
print("             Net Objective = {0:12.2f}".format(model.Value() - model.Cost()))
```

State Inventories

In [8]:

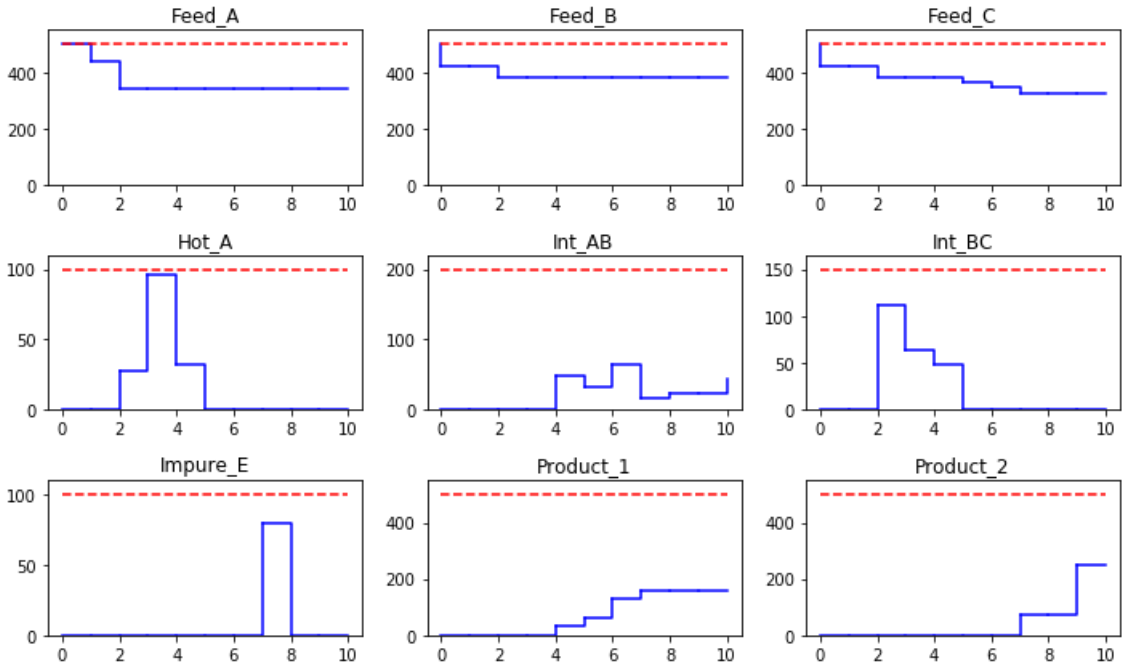
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display, HTML

pd.DataFrame([[model.S[s,t]() for s in STATES.keys()] for t in TIME], columns = STATES.
keys(), index = TIME)
```

Out[8]:

	Feed_A	Feed_B	Feed_C	Hot_A	Int_AB	Int_BC	Impure_E	Product_1	Product_2
0	500.0	420.0	420.0	0.0	0.0	0.0	0.0	0.0	0.0
1	440.0	420.0	420.0	0.0	0.0	0.0	0.0	0.0	0.0
2	340.0	380.0	380.0	28.0	0.0	112.0	0.0	0.0	0.0
3	340.0	380.0	380.0	96.0	0.0	64.0	0.0	0.0	0.0
4	340.0	380.0	380.0	32.0	48.0	48.0	0.0	32.0	0.0
5	340.0	380.0	364.0	0.0	32.0	0.0	0.0	64.0	0.0
6	340.0	380.0	348.0	0.0	64.0	0.0	0.0	128.0	0.0
7	340.0	380.0	324.0	0.0	16.0	0.0	80.0	160.0	72.0
8	340.0	380.0	324.0	0.0	24.0	0.0	0.0	160.0	72.0
9	340.0	380.0	324.0	0.0	24.0	0.0	0.0	160.0	252.0
10	340.0	380.0	324.0	0.0	44.0	0.0	0.0	160.0	252.0

In [9]:

```
plt.figure(figsize=(10,6))
for (s,idx) in zip(STATES.keys(),range(0,len(STATES.keys()))):
    plt.subplot(ceil(len(STATES.keys())/3),3,idx+1)
    tlast,ylast = 0,STATES[s]['initial']
    for (t,y) in zip(list(TIME),[model.S[s,t]() for t in TIME]):
        plt.plot([tlast,t,t],[ylast,ylast,y],'b')
        #plt.plot([tlast,t],[ylast,y],'b.',ms=10)
        tlast,ylast = t,y
    plt.ylim(0,1.1*C[s])
    plt.plot([0,H],[C[s],C[s]],'r--')
    plt.title(s)
plt.tight_layout()
```



Unit Assignment

In [10]:

```
UnitAssignment = pd.DataFrame({j:[None for t in TIME] for j in UNITS},index=TIME)

for t in TIME:
    for j in UNITS:
        for i in I[j]:
            for s in S_[i]:
                if t-p[i] >= 0:
                    if model.W[i,j,max(TIME[TIME <= t-p[i]])]() > 0:
                        UnitAssignment.loc[t,j] = None
        for i in I[j]:
            if model.W[i,j,t]() > 0:
                UnitAssignment.loc[t,j] = (i,model.B[i,j,t]())

UnitAssignment
```

Out[10]:

	Heater	Reactor_1	Reactor_2	Still
0	None	(Reaction_1, 80.0)	(Reaction_1, 80.0)	None
1	(Heating, 60.0)	None	None	None
2	(Heating, 100.0)	(Reaction_1, 80.0)	(Reaction_2, 80.0)	None
3	None	None	(Reaction_2, 80.0)	None
4	None	(Reaction_2, 80.0)	(Reaction_2, 80.0)	None
5	None	(Reaction_3, 80.0)	(Reaction_2, 80.0)	None
6	None	None	(Reaction_3, 80.0)	(Separation, 80.0)
7	None	(Reaction_3, 40.0)	(Reaction_3, 80.0)	None
8	None	None	None	(Separation, 200.0)
9	None	None	None	None
10	None	None	None	None

Unit Batch Inventories

In [11]:

```
pd.DataFrame([[model.Q[j,t]() for j in UNITS] for t in TIME], columns = UNITS, index =
TIME)
```

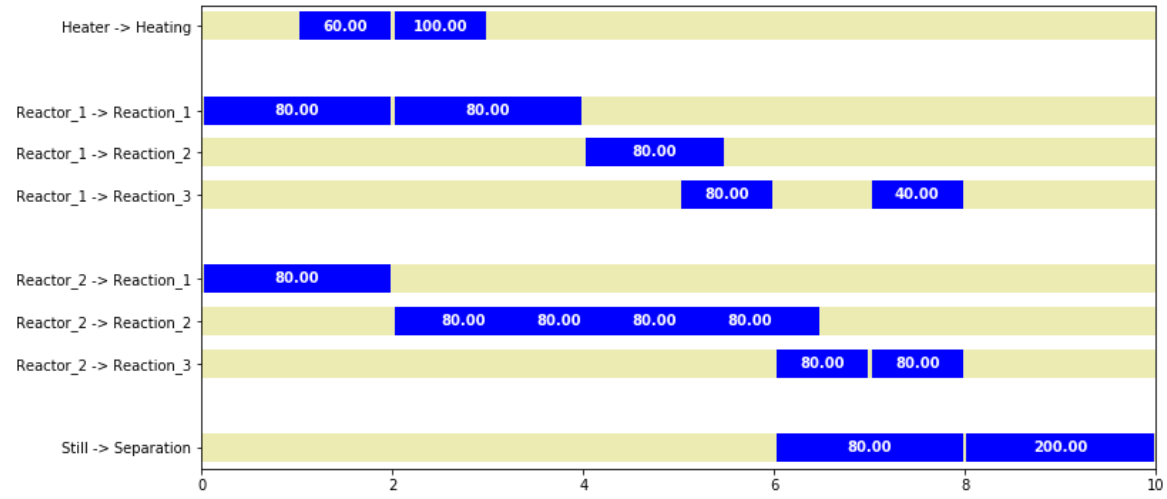
Out[11]:

	Still	Reactor_1	Reactor_2	Heater
0	0.0	80.0	80.0	0.0
1	0.0	80.0	80.0	60.0
2	0.0	80.0	80.0	100.0
3	0.0	80.0	160.0	0.0
4	0.0	80.0	160.0	0.0
5	0.0	160.0	160.0	0.0
6	80.0	0.0	160.0	0.0
7	8.0	40.0	80.0	0.0
8	200.0	0.0	0.0	0.0
9	20.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0

Gannt Chart

In [12]:

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))

gap = H/500
idx = 1
lbls = []
ticks = []
for j in sorted(UNITS):
    idx -= 1
    for i in sorted(I[j]):
        idx -= 1
        ticks.append(idx)
        lbls.append("{0:s} -> {1:s}".format(j,i))
        plt.plot([0,H],[idx,idx],lw=20,alpha=.3,color='y')
        for t in TIME:
            if model.W[i,j,t]() > 0:
                plt.plot([t+gap,t+p[i]-gap], [idx,idx],'b', lw=20, solid_capstyle='butt'
)
                txt = "{0:.2f}".format(model.B[i,j,t]())
                plt.text(t+p[i]/2, idx, txt, color='white', weight='bold', ha='center',
va='center')
plt.xlim(0,H)
plt.gca().set_yticks(ticks)
plt.gca().set_yticklabels(lbls);
```



Trace of Events and States

In [13]:

```
sep = '\n------------------------------------------------------------------------------
-----------\n'
print(sep)
print("Starting Conditions")
print("    Initial Inventories:")
for s in STATES.keys():
        print("        {0:10s}  {1:6.1f} kg".format(s,STATES[s]['initial']))

units = {j:{'assignment':'None', 't':0} for j in UNITS}

for t in TIME:
    print(sep)
    print("Time =",t,"hr")
    print("    Instructions:")
    for j in UNITS:
        units[j]['t'] += 1
        # transfer from unit to states
        for i in I[j]:
            for s in S_[i]:
                if t-P[(i,s)] >= 0:
                    amt = rho_[(i,s)]*model.B[i,j,max(TIME[TIME <= t - P[(i,s)]])]()
                    if amt > 0:
                        print("        Transfer", amt, "kg from", j, "to", s)
    for j in UNITS:
        # release units from tasks
        for i in I[j]:
            if t-p[i] >= 0:
                if model.W[i,j,max(TIME[TIME <= t-p[i]])]() > 0:
                    print("        Release", j, "from", i)
                    units[j]['assignment'] = 'None'
                    units[j]['t'] = 0
        # assign units to tasks
        for i in I[j]:
            if model.W[i,j,t]() > 0:
                print("        Assign", j, "with capacity", Bmax[(i,j)], "kg to task",i
"for",p[i],"hours")
                units[j]['assignment'] = i
                units[j]['t'] = 1
        # transfer from states to starting tasks
        for i in I[j]:
            for s in S[i]:
                amt = rho[(i,s)]*model.B[i,j,t]()
                if amt > 0:
                    print("        Transfer", amt,"kg from", s, "to", j)
    print("\n    Inventories are now:")
    for s in STATES.keys():
        print("        {0:10s}  {1:6.1f} kg".format(s,model.S[s,t]()))
    print("\n    Unit Assignments are now:")
    for j in UNITS:
        if units[j]['assignment'] != 'None':
            fmt = "        {0:s} performs the {1:s} task with a {2:.2f} kg batch for hc
ur {3:f} of {4:f}"
            i = units[j]['assignment']
            print(fmt.format(j,i,model.Q[j,t](),units[j]['t'],p[i]))

print(sep)
print('Final Conditions')
print("    Final Inventories:")
for s in STATES.keys():
        print("        {0:10s}  {1:6.1f} kg".format(s,model.S[s,H]()))
```

```
------------------------------------------------------------------------------
-

Starting Conditions
    Initial Inventories:
        Feed_A       500.0 kg
        Feed_B       500.0 kg
```

```
        Feed_B       500.0 kg
        Feed_C       500.0 kg
        Hot_A          0.0 kg
        Int_AB         0.0 kg
        Int_BC         0.0 kg
        Impure_E       0.0 kg
        Product_1      0.0 kg
        Product_2      0.0 kg

-------------------------------------------------------------------------------
-

Time = 0 hr
    Instructions:
        Assign Reactor_1 with capacity 80 kg to task Reaction_1 for 2 hours
        Transfer 40.0 kg from Feed_C to Reactor_1
        Transfer 40.0 kg from Feed_B to Reactor_1
        Assign Reactor_2 with capacity 80 kg to task Reaction_1 for 2 hours
        Transfer 40.0 kg from Feed_C to Reactor_2
        Transfer 40.0 kg from Feed_B to Reactor_2

    Inventories are now:
        Feed_A       500.0 kg
        Feed_B       420.0 kg
        Feed_C       420.0 kg
        Hot_A          0.0 kg
        Int_AB         0.0 kg
        Int_BC         0.0 kg
        Impure_E       0.0 kg
        Product_1      0.0 kg
        Product_2      0.0 kg

    Unit Assignments are now:
        Reactor_1 performs the Reaction_1 task with a 80.00 kg batch for hour 1.000000
of 2.000000
        Reactor_2 performs the Reaction_1 task with a 80.00 kg batch for hour 1.000000
of 2.000000

-------------------------------------------------------------------------------
-

Time = 1 hr
    Instructions:
        Assign Heater with capacity 100 kg to task Heating for 1 hours
        Transfer 60.0 kg from Feed_A to Heater

    Inventories are now:
        Feed_A       440.0 kg
        Feed_B       420.0 kg
        Feed_C       420.0 kg
        Hot_A          0.0 kg
        Int_AB         0.0 kg
        Int_BC         0.0 kg
        Impure_E       0.0 kg
        Product_1      0.0 kg
        Product_2      0.0 kg

    Unit Assignments are now:
        Reactor_1 performs the Reaction_1 task with a 80.00 kg batch for hour 2.000000
of 2.000000
        Reactor_2 performs the Reaction_1 task with a 80.00 kg batch for hour 2.000000
of 2.000000
        Heater performs the Heating task with a 60.00 kg batch for hour 1.000000 of 1.0
00000

-------------------------------------------------------------------------------
-

Time = 2 hr
```

```
    Instructions:
        Transfer 80.0 kg from Reactor_1 to Int_BC
        Transfer 80.0 kg from Reactor_2 to Int_BC
        Transfer 60.0 kg from Heater to Hot_A
        Release Reactor_1 from Reaction_1
        Assign Reactor_1 with capacity 80 kg to task Reaction_1 for 2 hours
        Transfer 40.0 kg from Feed_C to Reactor_1
        Transfer 40.0 kg from Feed_B to Reactor_1
        Release Reactor_2 from Reaction_1
        Assign Reactor_2 with capacity 80 kg to task Reaction_2 for 1.5 hours
        Transfer 32.0 kg from Hot_A to Reactor_2
        Transfer 48.0 kg from Int_BC to Reactor_2
        Release Heater from Heating
        Assign Heater with capacity 100 kg to task Heating for 1 hours
        Transfer 100.0 kg from Feed_A to Heater

    Inventories are now:
        Feed_A       340.0 kg
        Feed_B       380.0 kg
        Feed_C       380.0 kg
        Hot_A         28.0 kg
        Int_AB         0.0 kg
        Int_BC       112.0 kg
        Impure_E       0.0 kg
        Product_1      0.0 kg
        Product_2      0.0 kg

    Unit Assignments are now:
        Reactor_1 performs the Reaction_1 task with a 80.00 kg batch for hour 1.000000
of 2.000000
        Reactor_2 performs the Reaction_2 task with a 80.00 kg batch for hour 1.000000
of 1.500000
        Heater performs the Heating task with a 100.00 kg batch for hour 1.000000 of 1.
000000

---------------------------------------------------------------------------------
-

Time = 3 hr
    Instructions:
        Transfer 100.0 kg from Heater to Hot_A
        Assign Reactor_2 with capacity 80 kg to task Reaction_2 for 1.5 hours
        Transfer 32.0 kg from Hot_A to Reactor_2
        Transfer 48.0 kg from Int_BC to Reactor_2
        Release Heater from Heating

    Inventories are now:
        Feed_A       340.0 kg
        Feed_B       380.0 kg
        Feed_C       380.0 kg
        Hot_A         96.0 kg
        Int_AB         0.0 kg
        Int_BC        64.0 kg
        Impure_E       0.0 kg
        Product_1      0.0 kg
        Product_2      0.0 kg

    Unit Assignments are now:
        Reactor_1 performs the Reaction_1 task with a 80.00 kg batch for hour 2.000000
of 2.000000
        Reactor_2 performs the Reaction_2 task with a 160.00 kg batch for hour 1.000000
of 1.500000

---------------------------------------------------------------------------------
-

Time = 4 hr
    Instructions:
```

```
        Transfer 80.0 kg from Reactor_1 to Int_BC
        Transfer 32.0 kg from Reactor_2 to Product_1
        Transfer 48.0 kg from Reactor_2 to Int_AB
        Release Reactor_1 from Reaction_1
        Assign Reactor_1 with capacity 80 kg to task Reaction_2 for 1.5 hours
        Transfer 32.0 kg from Hot_A to Reactor_1
        Transfer 48.0 kg from Int_BC to Reactor_1
        Release Reactor_2 from Reaction_2
        Assign Reactor_2 with capacity 80 kg to task Reaction_2 for 1.5 hours
        Transfer 32.0 kg from Hot_A to Reactor_2
        Transfer 48.0 kg from Int_BC to Reactor_2

    Inventories are now:
        Feed_A       340.0 kg
        Feed_B       380.0 kg
        Feed_C       380.0 kg
        Hot_A         32.0 kg
        Int_AB        48.0 kg
        Int_BC        48.0 kg
        Impure_E       0.0 kg
        Product_1     32.0 kg
        Product_2      0.0 kg

    Unit Assignments are now:
        Reactor_1 performs the Reaction_2 task with a 80.00 kg batch for hour 1.000000
of 1.500000
        Reactor_2 performs the Reaction_2 task with a 160.00 kg batch for hour 1.000000
of 1.500000

---------------------------------------------------------------------------------------
-

Time = 5 hr
    Instructions:
        Transfer 32.0 kg from Reactor_2 to Product_1
        Transfer 48.0 kg from Reactor_2 to Int_AB
        Assign Reactor_1 with capacity 80 kg to task Reaction_3 for 1 hours
        Transfer 16.0 kg from Feed_C to Reactor_1
        Transfer 64.0 kg from Int_AB to Reactor_1
        Release Reactor_2 from Reaction_2
        Assign Reactor_2 with capacity 80 kg to task Reaction_2 for 1.5 hours
        Transfer 32.0 kg from Hot_A to Reactor_2
        Transfer 48.0 kg from Int_BC to Reactor_2

    Inventories are now:
        Feed_A       340.0 kg
        Feed_B       380.0 kg
        Feed_C       364.0 kg
        Hot_A          0.0 kg
        Int_AB        32.0 kg
        Int_BC         0.0 kg
        Impure_E       0.0 kg
        Product_1     64.0 kg
        Product_2      0.0 kg

    Unit Assignments are now:
        Reactor_1 performs the Reaction_3 task with a 160.00 kg batch for hour 1.000000
of 1.000000
        Reactor_2 performs the Reaction_2 task with a 160.00 kg batch for hour 1.000000
of 1.500000

---------------------------------------------------------------------------------------
-

Time = 6 hr
    Instructions:
        Transfer 32.0 kg from Reactor_1 to Product_1
        Transfer 48.0 kg from Reactor_1 to Int_AB
        Transfer 80.0 kg from Reactor_1 to Impure_E
```

```
        Transfer 32.0 kg from Reactor_2 to Product_1
        Transfer 48.0 kg from Reactor_2 to Int_AB
        Assign Still with capacity 200 kg to task Separation for 2 hours
        Transfer 80.0 kg from Impure_E to Still
        Release Reactor_1 from Reaction_2
        Release Reactor_1 from Reaction_3
        Release Reactor_2 from Reaction_2
        Assign Reactor_2 with capacity 80 kg to task Reaction_3 for 1 hours
        Transfer 16.0 kg from Feed_C to Reactor_2
        Transfer 64.0 kg from Int_AB to Reactor_2

    Inventories are now:
        Feed_A       340.0 kg
        Feed_B       380.0 kg
        Feed_C       348.0 kg
        Hot_A          0.0 kg
        Int_AB        64.0 kg
        Int_BC         0.0 kg
        Impure_E       0.0 kg
        Product_1    128.0 kg
        Product_2      0.0 kg

    Unit Assignments are now:
        Still performs the Separation task with a 80.00 kg batch for hour 1.000000 of 2
.000000
        Reactor_2 performs the Reaction_3 task with a 160.00 kg batch for hour 1.000000
of 1.000000

------------------------------------------------------------------------------------
-

Time = 7 hr
    Instructions:
        Transfer 72.0 kg from Still to Product_2
        Transfer 32.0 kg from Reactor_2 to Product_1
        Transfer 48.0 kg from Reactor_2 to Int_AB
        Transfer 80.0 kg from Reactor_2 to Impure_E
        Assign Reactor_1 with capacity 80 kg to task Reaction_3 for 1 hours
        Transfer 8.0 kg from Feed_C to Reactor_1
        Transfer 32.0 kg from Int_AB to Reactor_1
        Release Reactor_2 from Reaction_2
        Release Reactor_2 from Reaction_3
        Assign Reactor_2 with capacity 80 kg to task Reaction_3 for 1 hours
        Transfer 16.0 kg from Feed_C to Reactor_2
        Transfer 64.0 kg from Int_AB to Reactor_2

    Inventories are now:
        Feed_A       340.0 kg
        Feed_B       380.0 kg
        Feed_C       324.0 kg
        Hot_A          0.0 kg
        Int_AB        16.0 kg
        Int_BC         0.0 kg
        Impure_E      80.0 kg
        Product_1    160.0 kg
        Product_2     72.0 kg

    Unit Assignments are now:
        Still performs the Separation task with a 8.00 kg batch for hour 2.000000 of 2.
000000
        Reactor_1 performs the Reaction_3 task with a 40.00 kg batch for hour 1.000000
of 1.000000
        Reactor_2 performs the Reaction_3 task with a 80.00 kg batch for hour 1.000000
of 1.000000

------------------------------------------------------------------------------------
-
```

```
Time = 8 hr
    Instructions:
        Transfer 8.0 kg from Still to Int_AB
        Transfer 40.0 kg from Reactor_1 to Impure_E
        Transfer 80.0 kg from Reactor_2 to Impure_E
        Release Still from Separation
        Assign Still with capacity 200 kg to task Separation for 2 hours
        Transfer 200.0 kg from Impure_E to Still
        Release Reactor_1 from Reaction_3
        Release Reactor_2 from Reaction_3

    Inventories are now:
        Feed_A       340.0 kg
        Feed_B       380.0 kg
        Feed_C       324.0 kg
        Hot_A          0.0 kg
        Int_AB        24.0 kg
        Int_BC         0.0 kg
        Impure_E       0.0 kg
        Product_1    160.0 kg
        Product_2     72.0 kg

    Unit Assignments are now:
        Still performs the Separation task with a 200.00 kg batch for hour 1.000000 of
2.000000

-------------------------------------------------------------------------------
-

Time = 9 hr
    Instructions:
        Transfer 180.0 kg from Still to Product_2

    Inventories are now:
        Feed_A       340.0 kg
        Feed_B       380.0 kg
        Feed_C       324.0 kg
        Hot_A          0.0 kg
        Int_AB        24.0 kg
        Int_BC         0.0 kg
        Impure_E       0.0 kg
        Product_1    160.0 kg
        Product_2    252.0 kg

    Unit Assignments are now:
        Still performs the Separation task with a 20.00 kg batch for hour 2.000000 of 2
.000000

-------------------------------------------------------------------------------
-

Time = 10 hr
    Instructions:
        Transfer 20.0 kg from Still to Int_AB
        Release Still from Separation

    Inventories are now:
        Feed_A       340.0 kg
        Feed_B       380.0 kg
        Feed_C       324.0 kg
        Hot_A          0.0 kg
        Int_AB        44.0 kg
        Int_BC         0.0 kg
        Impure_E       0.0 kg
        Product_1    160.0 kg
        Product_2    252.0 kg

    Unit Assignments are now:
```

```
------------------------------------------------------------------------------------
-

Final Conditions
    Final Inventories:
        Feed_A       340.0 kg
        Feed_B       380.0 kg
        Feed_C       324.0 kg
        Hot_A          0.0 kg
        Int_AB        44.0 kg
        Int_BC         0.0 kg
        Impure_E       0.0 kg
        Product_1    160.0 kg
        Product_2    252.0 kg
```

In [ ]:

Chapter 5. Simulation

5.1 Response of a First Order System to Step and Square Wave Inputs

In [0]:

```
!pip install -q pyomo
```

First-Order Differential Equation with Constant Input

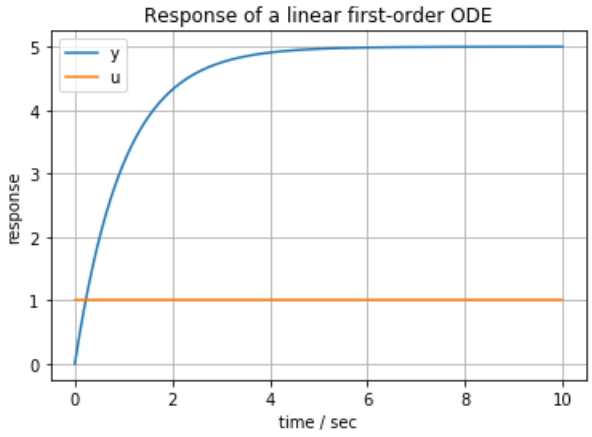
The following cell simulates the response of a first-order linear model in the form

$$\tau\frac{dy}{dt} + y = Ku(t)$$

where τ and K are model parameters, and $u(t) = 1$ is an external process input.

In [2]:

```
% matplotlib inline
from pyomo.environ import *
from pyomo.dae import *
import matplotlib.pyplot as plt

tfinal = 10
tau = 1
K = 5

# define u(t)
u = lambda t: 1

# create a model object
model = ConcreteModel()

# define the independent variable
model.t = ContinuousSet(bounds=(0, tfinal))

# define the dependent variables
model.y = Var(model.t)
model.dydt = DerivativeVar(model.y)

# fix the initial value of y
model.y[0].fix(0)

# define the differential equation as a constraint
model.ode = Constraint(model.t, rule=lambda model, t: tau*model.dydt[t] + model.y[t] ==
K*u(t))

# simulation using scipy integrators
tsim, profiles = Simulator(model, package='scipy').simulate(numpoints=1000)

fig, ax = plt.subplots(1, 1)
ax.plot(tsim, profiles, label='y')
ax.plot(tsim, [u(t) for t in tsim], label='u')
ax.set_xlabel('time / sec')
ax.set_ylabel('response')
ax.set_title('Response of a linear first-order ODE')
ax.legend()
ax.grid(True)
```

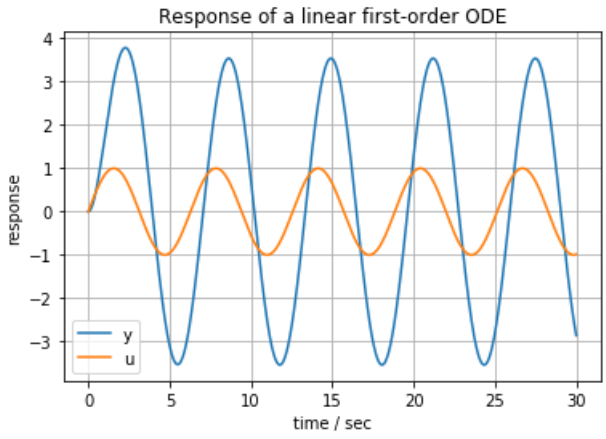


Encapsulating into a Function

In following cells we would like to explore the response of a first order system to changes in parameters and input functions. To facilitate this study, the next cell encapsulates the simulation into a function that can be called with different parameter values and input functions.

In [3]:

```
% matplotlib inline
from pyomo.environ import *
from pyomo.dae import *
import matplotlib.pyplot as plt

def first_order(K=1, tau=1, tfinal=1, u=lambda t: 1):
    model = ConcreteModel()
    model.t = ContinuousSet(bounds=(0, tfinal))
    model.y = Var(model.t)
    model.dydt = DerivativeVar(model.y)
    model.y[0].fix(0)
    model.ode = Constraint(model.t, rule=lambda model, t:
                           tau*model.dydt[t] + model.y[t] == K*u(t))

    tsim, profiles = Simulator(model, package='scipy').simulate(numpoints=1000)

    fig, ax = plt.subplots(1, 1)
    ax.plot(tsim, profiles, label='y')
    ax.plot(tsim, [u(t) for t in tsim], label='u')
    ax.set_xlabel('time / sec')
    ax.set_ylabel('response')
    ax.set_title('Response of a linear first-order ODE')
    ax.legend()
    ax.grid(True)

first_order(5, 1, 30, sin)
```



Analytical Approximation to a Step Input

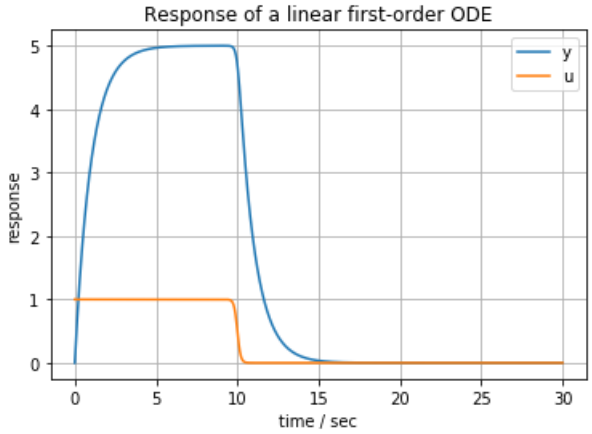
The response to a step change is a common test giving insight into the dynamics of a given system. An infinitely differentiable approximation to a step change is given by the *Butterworth function* $b_n(t)$

$$b_n(t) = \frac{1}{1 + (\frac{t}{c})^n}$$

where n is the order of a approximation, and c is value of t where the step change occurs.

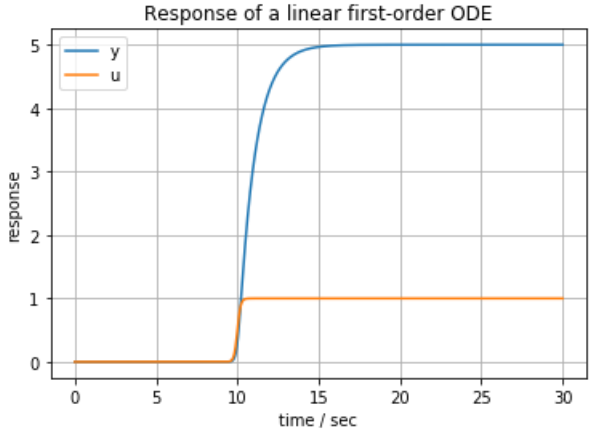
In [4]:

```
u = lambda t: 1/(1 + (t/10)**100)

first_order(5, 1, 30, u)
```



In [5]:

```
u = lambda t: 1 - 1/(1 + (t/10)**100)

first_order(5, 1, 30, u)
```



Analytical Approximation to a Square Wave Input

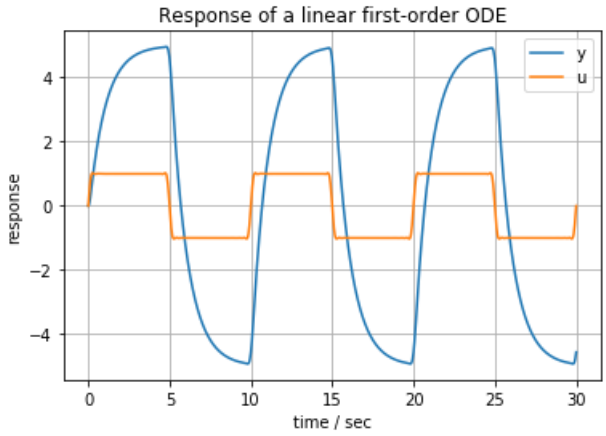
An analytical approximation to a square wave with frequency f is given by

$$\frac{4}{\pi} \sum_{k=1,3,5,\ldots}^{N} \frac{sin(k\pi/N)}{k\pi/N} \frac{sin(2\pi ft)}{k}$$

where the first term is the *Lanczos* sigma factor designed to suppress the Gibb's phenomenon associated with Fourier series approximations.

In [6]:

```
from math import pi
def square(t, f=1, N=31):
    return (4/pi)*sum((N*sin(k*pi/N)/k/pi)*sin(2*k*f*pi*t)/k for k in range(1, N+1,2))

u = lambda t: square(t, 0.1)

first_order(5, 1, 30, u)
```



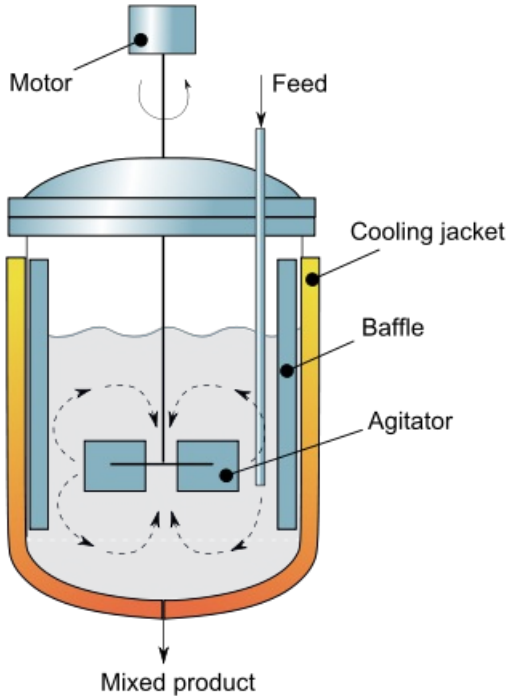
5.2 Exothermic CSTR

In [ ]:

```
!pip install -q pyomo
```

Description

This example is intended as an introduction to the nonlinear dynamics of an exothermic continuous stirred-tank reactor. The example has been studied by countless researchers and students since the pioneering work of Amundson and Aris in the 1950's. The particular formulation and parameter values described below are taken from example 2.5 from Seborg, Edgar, Mellichamp and Doyle (SEMD).



(Diagram By Daniele Pugliesi - Own work, CC BY-SA 3.0, Link)

Arrehenius Law Kinetics for a First-Order Reaction

We assume the kinetics are dominated by a single first order reaction

$$A \xrightarrow{kc_A} \text{Products}$$

The reaction rate per unit volume is modeled as the product kc_A where c_A is the concentration of A. The rate constant $k(T)$ is a increases with temperature following the Arrehenius law

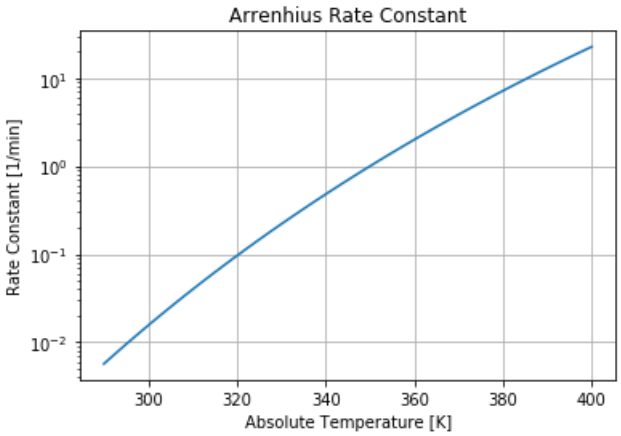
$$k(t) = k_0 e^{-\frac{E_a}{RT}}$$

E_a is the activation energy, R is the gas constant, T is absolute temperature, and k_0 is the pre-exponential factor.

We can see the strong temperature dependence by plotting $k(T)$ versus temperature over typical operating conditions.

In [2]:

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

Ea  = 72750     # activation energy J/gmol
R   = 8.314     # gas constant J/gmol/K
k0  = 7.2e10    # Arrhenius rate constant 1/min

T = np.linspace(290,400)

# Arrhenius rate expression
def k(T):
    return k0*np.exp(-Ea/R/T)

plt.semilogy(T,k(T))
plt.xlabel('Absolute Temperature [K]')
plt.ylabel('Rate Constant [1/min]')
plt.title('Arrenhius Rate Constant')
plt.grid(True)
```



This graph shows the reaction rate changes by three orders of magnitude over the range of possible operating temperatures. Because an exothermic reaction releases heat faster at higher temperatures, there is a positive feedback that can potentially result in unstable process behavior.

Modeling and Parameter Values

Mathematical Model

The model consists of mole and energy balances on the contents of the well-mixed reactor.

$$V\frac{dc_A}{dt} = q(c_{Ai} - c_A) - Vkc_A$$
$$V\rho C_p\frac{dT}{dt} = wC_p(T_i - T) + (-\Delta H_R)Vkc_A + UA(T_c - T)$$

which are the equations that will be integrated below.

Quantity	Symbol	Value	Units	Comments
Activation Energy	E_a	72,750	J/gmol	
Arrehnius pre-exponential	k_0	7.2 x 10^{10}	1/min	
Gas Constant	R	8.314	J/gmol/K	
Reactor Volume	V	100	liters	
Density	ρ	1000	g/liter	
Heat Capacity	C_p	0.239	J/g/K	
Enthalpy of Reaction	ΔH_r	-50,000	J/gmol	
Heat Transfer Coefficient	UA	50,000	J/min/K	
Feed flowrate	q	100	liters/min	
Feed concentration	$c_{A,f}$	1.0	gmol/liter	
Feed temperature	T_f	350	K	
Initial concentration	$c_{A,0}$	0.5	gmol/liter	
Initial temperature	T_0	350	K	
Coolant temperature	T_c	300	K	Primary Manipulated Variable

Pyomo Model

In [ ]:

```
from pyomo.environ import *
from pyomo.dae import *
from pyomo.dae.simulator import Simulator

Ea  = 72750     # activation energy J/gmol
R   = 8.314     # gas constant J/gmol/K
k0  = 7.2e10    # Arrhenius rate constant 1/min
V   = 100.0     # Volume [L]
rho = 1000.0    # Density [g/L]
Cp  = 0.239     # Heat capacity [J/g/K]
dHr = -5.0e4    # Enthalpy of reaction [J/mol]
UA  = 5.0e4     # Heat transfer [J/min/K]
q   = 100.0     # Flowrate [L/min]
cAi = 1.0       # Inlet feed concentration [mol/L]
Ti  = 350.0     # Inlet feed temperature [K]
cA0 = 0.5       # Initial concentration [mol/L]
T0  = 350.0     # Initial temperature [K]
Tc  = 300.0     # Coolant temperature [K]

def cstr(cA0 = 0.5, T0 = 350.0):
    m     = ConcreteModel()
    m.t   = ContinuousSet(bounds=(0.0, 10.0))
    m.cA  = Var(m.t)
    m.T   = Var(m.t)
    m.dcA = DerivativeVar(m.cA)
    m.dT  = DerivativeVar(m.T)

    # Setting the initial conditions
    m.cA[0.0] = cA0
    m.T[0.0]  = T0

    k = lambda T: k0*exp(-Ea/R/T)
    m.ode1 = Constraint(m.t, rule=lambda m, t:
        V*m.dcA[t] == q*(cAi - m.cA[t]) - V*k(m.T[t])*m.cA[t])
    m.ode2 = Constraint(m.t, rule=lambda m, t:
        V*rho*Cp*m.dT[t] == q*rho*Cp*(Ti - m.T[t]) + (-dHr)*V*k(m.T[t])*m.cA[t] + UA*(Tc
- m.T[t]))

    return m
```

Simulation and Visualization

Visualization Function

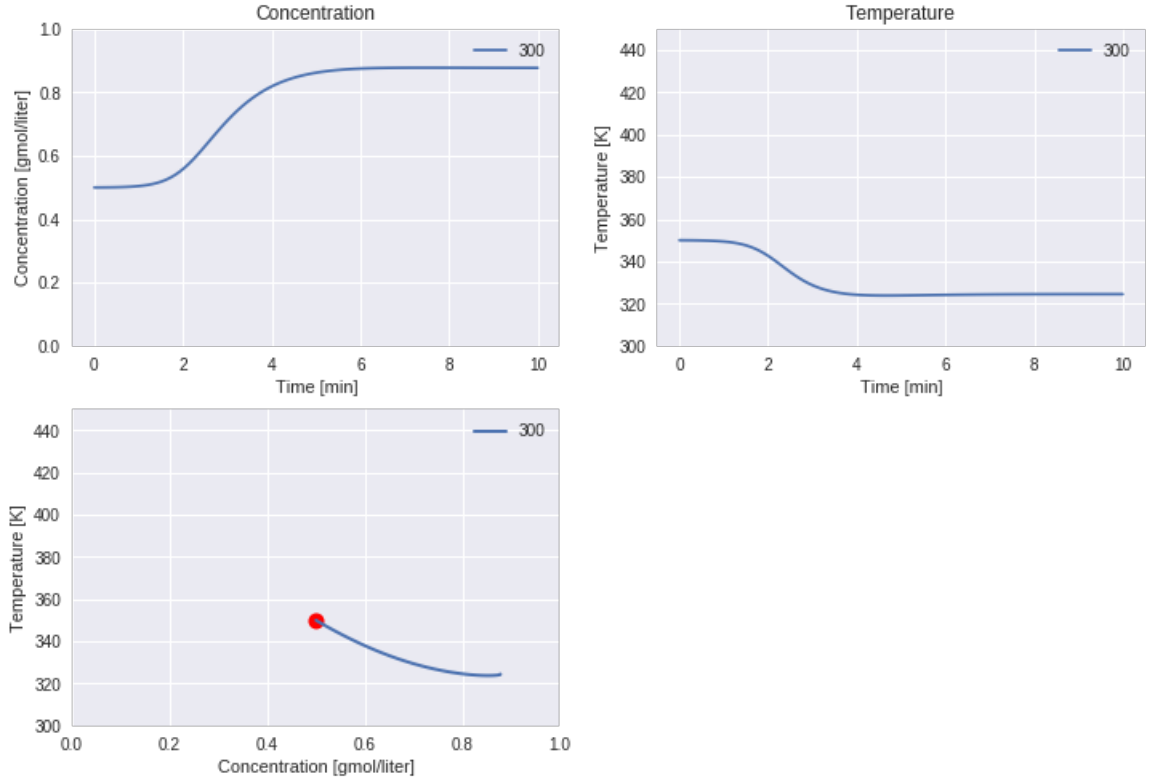
In [ ]:

```
%matplotlib inline
import matplotlib.pyplot as plt

# visualization function plots concentration and temperature
def cstr_plot(t, y, ax=[]):
    if len(ax) == 0:
        fig = plt.figure(figsize=(12,8))

        ax1 = plt.subplot(2,2,1)
        plt.xlabel('Time [min]')
        plt.ylabel('Concentration [gmol/liter]')
        plt.title('Concentration')
        plt.ylim(0,1)

        ax2 = plt.subplot(2,2,2);
        plt.xlabel('Time [min]')
        plt.ylabel('Temperature [K]');
        plt.title('Temperature')
        plt.ylim(300,450)

        ax3 = plt.subplot(2,2,3);
        plt.xlabel('Concentration [gmol/liter]')
        plt.ylabel('Temperature [K]');
        plt.xlim(0,1)
        plt.ylim(300,450)
    else:
        ax1, ax2, ax3 = ax
    ax1.plot(t, y[:,0], label=str(Tc))
    ax1.legend()
    ax2.plot(t, y[:,1], label=str(Tc))
    ax2.legend()
    ax3.plot(y[0,0],y[0,1],'r.',ms=20)
    ax3.plot(y[:,0],y[:,1], lw=2, label=str(Tc))
    ax3.legend()
    return [ax1, ax2, ax3]
```

Simulation

In [7]:

```
Tc = 300
tsim, profiles = Simulator(cstr(), package='scipy').simulate(numpoints=100)
cstr_plot(tsim, profiles);
```

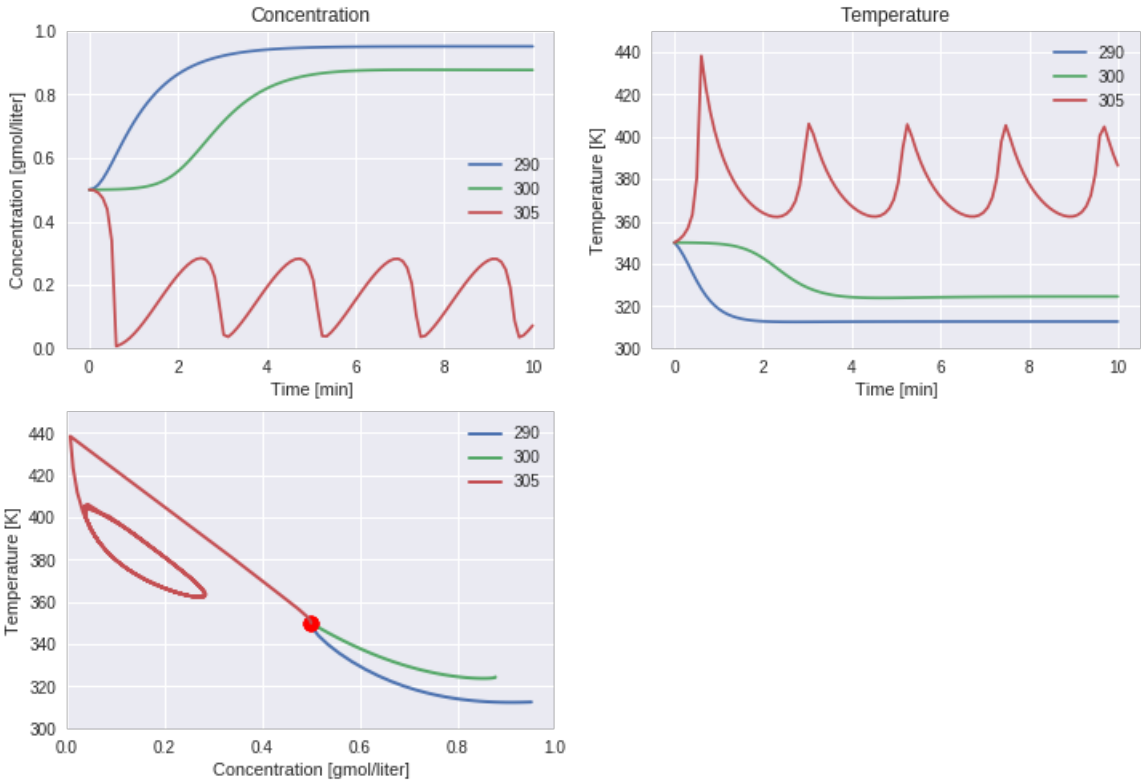


Effect of Cooling Temperature

The primary means of controlling the reactoris through temperature of the cooling water jacket. The next calculations explore the effect of plus or minus change of 5 K in cooling water temperature on reactor behavior. These simulations reproduce the behavior shown in Example 2.5 SEMD.

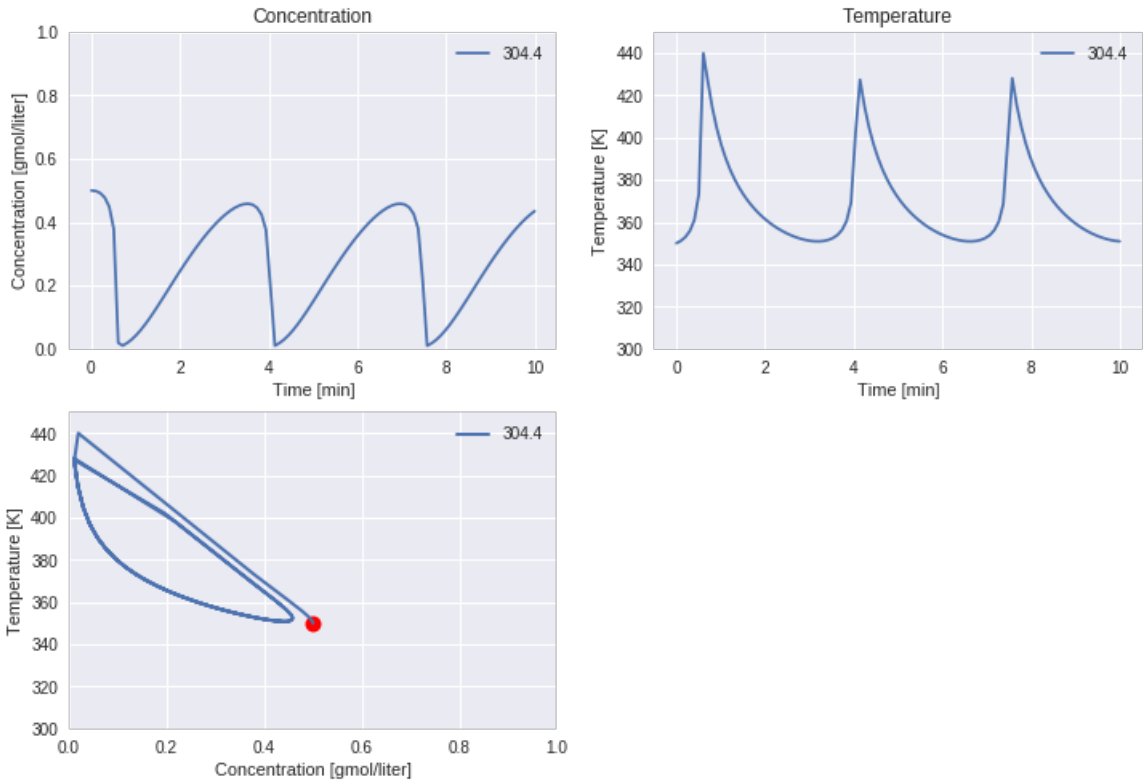
In [ ]:

```
ax = []
for Tc in [290, 300, 305]:
    tsim, profiles = Simulator(cstr(), package='scipy').simulate(numpoints=100)
    ax  = cstr_plot(tsim, profiles, ax)
```



In [32]:

```
#@title CSTR Simulation { run: "auto", vertical-output: true }
T_cooling = 304.4 #@param {type:"slider", min:290, max:305, step:0.1}
ax = []
Tc = T_cooling
tsim, profiles = Simulator(cstr(), package='scipy').simulate(numpoints=100)
ax  = cstr_plot(tsim, profiles, ax);
```



5.3 Transient Heat Conduction in Various Geometries

Rescaling

We'll assume the thermal conductivity k is a constant, and define thermal diffusivity in the conventional way

$$\alpha = \frac{k}{\rho C_p}$$

We will further assume symmetry with respect to all spatial coordinates except r where r extends from $-R$ to $+R$. The boundary conditions are

$$\begin{aligned} T(t,R) &= T_\infty & \forall t > 0 \\ \nabla T(t,0) &= 0 & \forall t \geq 0 \end{aligned}$$

where we have assumed symmetry with respect to r and uniform initial conditions $T(0,r) = T_0$ for all $0 \leq r \leq R$. Following standard scaling procedures, we introduce the dimensionless variables

$$\begin{aligned} T' &= \frac{T - T_0}{T_\infty - T_0} \\ r' &= \frac{r}{R} \\ t' &= t\frac{\alpha}{R^2} \end{aligned}$$

Dimensionless Model

Under these conditions the problem reduces to

$$\frac{\partial T'}{\partial t'} = \nabla^2 T'$$

with auxiliary conditions

$$\begin{aligned} T'(0,r') &= 0 & \forall 0 \leq r' \leq 1 \\ T'(t',1) &= 1 & \forall t' > 0 \\ \nabla T'(t',0) &= 0 & \forall t' \geq 0 \end{aligned}$$

which we can specialize to specific geometries.

Preliminary Code

In [ ]:

```
!pip install -q pyomo
!wget -N -q "https://ampl.com/dl/open/ipopt/ipopt-linux64.zip"
!unzip -o -q ipopt-linux64

ipopt_executable='/content/ipopt'
```

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D

def model_plot(m):
    r = sorted(m.r)
    t = sorted(m.t)

    rgrid = np.zeros((len(t), len(r)))
    tgrid = np.zeros((len(t), len(r)))
    Tgrid = np.zeros((len(t), len(r)))

    for i in range(0, len(t)):
        for j in range(0, len(r)):
            rgrid[i,j] = r[j]
            tgrid[i,j] = t[i]
            Tgrid[i,j] = m.T[t[i], r[j]].value

    fig = plt.figure(figsize=(10,6))
    ax = fig.add_subplot(1, 1, 1, projection='3d')
    ax.set_xlabel('Distance r')
    ax.set_ylabel('Time t')
    ax.set_zlabel('Temperature T')
    p = ax.plot_wireframe(rgrid, tgrid, Tgrid)
```

Planar Coordinates

Suppressing the prime notation, for a slab geometry the model specializes to

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial r^2}$$

with auxiliary conditions

$$\begin{aligned} T(0,r) &= 0 \qquad \forall 0 \le r \le 1 \\ T(t,1) &= 1 \qquad \forall t > 0 \\ \frac{\partial T}{\partial r}(t,0) &= 0 \qquad \forall t \ge 0 \end{aligned}$$

In [11]:

```
!ls -a /content
```

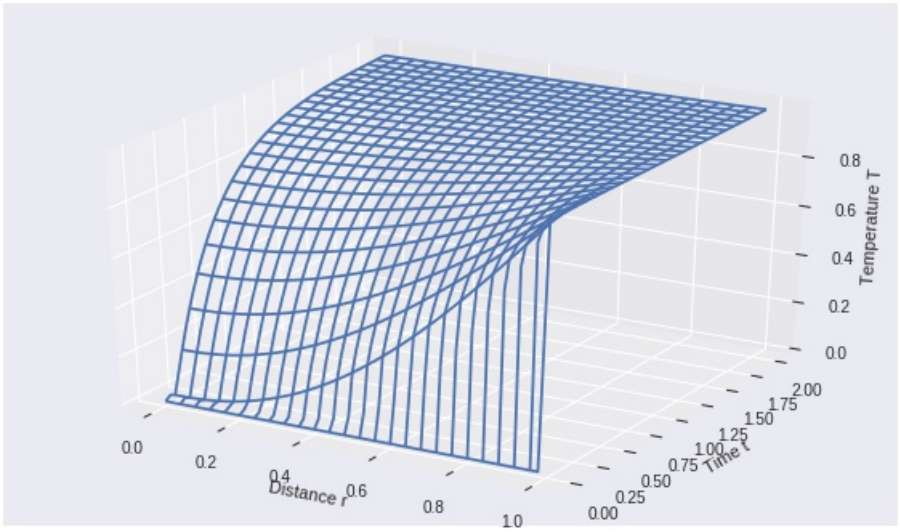
```
.  ..  coin-license.txt  .config  ipopt  ipopt-linux64.zip  sample_data
```

In [12]:

```
from pyomo.environ import *
from pyomo.dae import *

m = ConcreteModel()

m.r = ContinuousSet(bounds=(0,1))
m.t = ContinuousSet(bounds=(0,2))

m.T = Var(m.t, m.r)

m.dTdt   = DerivativeVar(m.T, wrt=m.t)
m.dTdr   = DerivativeVar(m.T, wrt=m.r)
m.d2Tdr2 = DerivativeVar(m.T, wrt=(m.r, m.r))

@m.Constraint(m.t, m.r)
def pde(m, t, r):
    if t == 0:
        return Constraint.Skip
    if r == 0 or r == 1:
        return Constraint.Skip
    return m.dTdt[t,r] == m.d2Tdr2[t,r]

m.obj = Objective(expr=1)

m.ic  = Constraint(m.r, rule=lambda m, r:    m.T[0,r] == 0 if r > 0 and r < 1 else
Constraint.Skip)
m.bc1 = Constraint(m.t, rule=lambda m, t:    m.T[t,1] == 1)
m.bc2 = Constraint(m.t, rule=lambda m, t: m.dTdr[t,0] == 0)

TransformationFactory('dae.finite_difference').apply_to(m, nfe=50, scheme='FORWARD', wr
t=m.r)
TransformationFactory('dae.finite_difference').apply_to(m, nfe=50, scheme='FORWARD', wr
t=m.t)
SolverFactory('ipopt', executable=ipopt_executable).solve(m, tee=True).write()
model_plot(m)
```

```
Ipopt 3.12.8:

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

This is Ipopt version 3.12.8, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

Number of nonzeros in equality constraint Jacobian...:    30347
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        0

Total number of variables............................:    10299
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:    10200
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  1.0000000e+00 1.00e+00 0.00e+00  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  1.0000000e+00 1.50e-12 2.50e-01  -1.7 2.50e+03  -4.0 1.00e+00 1.00e+00h  1
   2  1.0000000e+00 1.82e-12 1.53e-10  -1.7 8.16e-10  -4.5 1.00e+00 1.00e+00h  1

Number of Iterations....: 2
```

```
Number of iterations....: 2

                                   (scaled)                 (unscaled)
Objective...............:   1.0000000000000000e+00    1.0000000000000000e+00
Dual infeasibility......:   1.5268200213081659e-10    1.5268200213081659e-10
Constraint violation....:   3.6364522504328498e-14    1.8182261252164267e-12
Complementarity.........:   0.0000000000000000e+00    0.0000000000000000e+00
Overall NLP error.......:   1.5268200213081659e-10    1.5268200213081659e-10


Number of objective function evaluations             = 3
Number of objective gradient evaluations             = 3
Number of equality constraint evaluations            = 3
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 3
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 2
Total CPU secs in IPOPT (w/o function evaluations)   =      0.207
Total CPU secs in NLP function evaluations           =      0.003

EXIT: Optimal Solution Found.
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 10200
  Number of variables: 10299
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: Ipopt 3.12.8\x3a Optimal Solution Found
  Termination condition: optimal
  Id: 0
  Error rc: 0
  Time: 0.30750346183776855
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```



Cylindrical Coordinates

Suppressing the prime notation, for a cylindrical geometry the model specializes to

$$\frac{\partial T}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right)$$

Expanding,

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial t^2} + \frac{1}{r}\frac{\partial T}{\partial r}$$

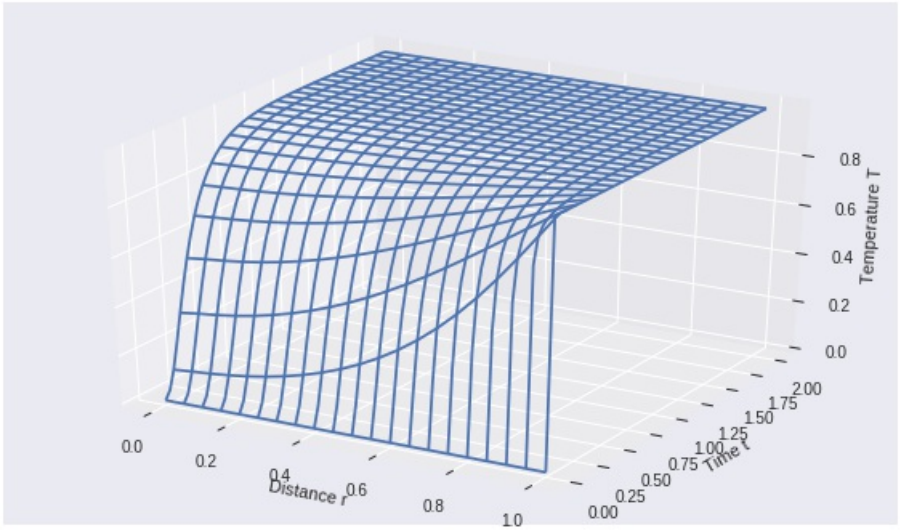
with auxiliary conditions

$$\begin{aligned} T(0,r) &= 0 \qquad \forall 0 \leq r \leq 1 \\ T(t,1) &= 1 \qquad \forall t > 0 \\ \frac{\partial T}{\partial r}(t,0) &= 0 \qquad \forall t \geq 0 \end{aligned}$$

In [13]:

```
from pyomo.environ import *
from pyomo.dae import *

m = ConcreteModel()

m.r = ContinuousSet(bounds=(0,1))
m.t = ContinuousSet(bounds=(0,2))

m.T = Var(m.t, m.r)

m.dTdt   = DerivativeVar(m.T, wrt=m.t)
m.dTdr   = DerivativeVar(m.T, wrt=m.r)
m.d2Tdr2 = DerivativeVar(m.T, wrt=(m.r, m.r))

m.pde = Constraint(m.t, m.r, rule=lambda m, t, r: m.dTdt[t,r] == m.d2Tdr2[t,r] + (1/r)*m.
dTdr[t,r]
        if r > 0 and r < 1 and t > 0 else Constraint.Skip)

m.ic  = Constraint(m.r, rule=lambda m, r:    m.T[0,r] == 0)
m.bc1 = Constraint(m.t, rule=lambda m, t:    m.T[t,1] == 1 if t > 0 else Constraint.Skip
)
m.bc2 = Constraint(m.t, rule=lambda m, t: m.dTdr[t,0] == 0)

TransformationFactory('dae.finite_difference').apply_to(m, nfe=20, wrt=m.r, scheme='CEN
TRAL')
TransformationFactory('dae.finite_difference').apply_to(m, nfe=50, wrt=m.t, scheme='BAC
KWARD')
SolverFactory('ipopt', executable=ipopt_executable).solve(m).write()

model_plot(m)
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 4060
  Number of variables: 4110
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: Ipopt 3.12.8\x3a Optimal Solution Found
  Termination condition: optimal
  Id: 0
  Error rc: 0
  Time: 0.16563725471496582
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```



Spherical Coordinates

Suppressing the prime notation, for a cylindrical geometry the model specializes to

$$\frac{\partial T}{\partial t} = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial T}{\partial r}\right)$$

Expanding,

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial t^2} + \frac{2}{r}\frac{\partial T}{\partial r}$$

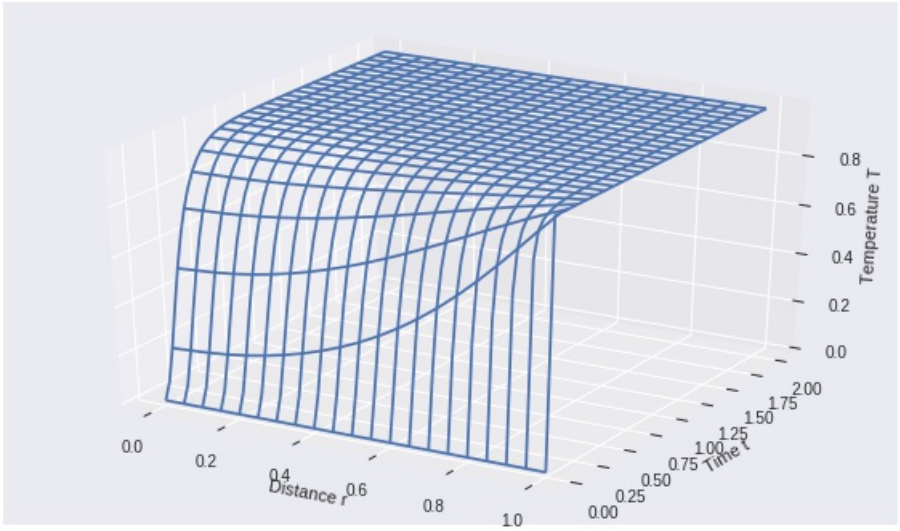
with auxiliary conditions

$$\begin{aligned} T(0,r) &= 0 \qquad \forall 0 \le r \le 1 \\ T(t,1) &= 1 \qquad \forall t > 0 \\ \frac{\partial T}{\partial r}(t,0) &= 0 \qquad \forall t \ge 0 \end{aligned}$$

In [14]:

```
from pyomo.environ import *
from pyomo.dae import *

m = ConcreteModel()

m.r = ContinuousSet(bounds=(0,1))
m.t = ContinuousSet(bounds=(0,2))

m.T = Var(m.t, m.r)

m.dTdt   = DerivativeVar(m.T, wrt=m.t)
m.dTdr   = DerivativeVar(m.T, wrt=m.r)
m.d2Tdr2 = DerivativeVar(m.T, wrt=(m.r, m.r))

m.pde = Constraint(m.t, m.r, rule=lambda m, t, r: m.dTdt[t,r] == m.d2Tdr2[t,r] + (2/r)*m.
dTdr[t,r]
        if r > 0 and r < 1 and t > 0 else Constraint.Skip)

m.ic  = Constraint(m.r, rule=lambda m, r:    m.T[0,r] == 0)
m.bc1 = Constraint(m.t, rule=lambda m, t:    m.T[t,1] == 1 if t > 0 else Constraint.Skip
)
m.bc2 = Constraint(m.t, rule=lambda m, t: m.dTdr[t,0] == 0)

TransformationFactory('dae.finite_difference').apply_to(m, nfe=20, wrt=m.r, scheme='CEN
TRAL')
TransformationFactory('dae.finite_difference').apply_to(m, nfe=50, wrt=m.t, scheme='BAC
KWARD')
SolverFactory('ipopt', executable=ipopt_executable).solve(m).write()

model_plot(m)
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 4060
  Number of variables: 4110
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: Ipopt 3.12.8\x3a Optimal Solution Found
  Termination condition: optimal
  Id: 0
  Error rc: 0
  Time: 0.15887713432312012
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```



In [ ]:

Chapter 6. Differential-Algebraic Equations

6.1 Unconstrained Scalar Optimization

Application: Maximizing Production of a Reaction Intermediate

A desired product B is produced as intermediate in a series reaction

$$A \xrightarrow{k_A} B \xrightarrow{k_B} C$$

where A is a raw material and C is a undesired by-product. The reaction operates at temperature where the rate constants are $k_A = 0.5\ \text{min}^{-1}$ and $k_A = 0.1\ \text{min}^{-1}$. The raw material is available as a solution with concenration $C_{A,f} = 2.0\ \text{moles}/\text{liter}$.

A 100 liter tank is avialable to run the reaction. Below we will answer the following questions:

1. If the goal is obtain the maximum possible concentration of B, and the tank is operated as a continuous stirred tank reactor, what should be the flowrate?
2. What is the production rate of B at maximum concentration?

Mathematical Model for a Continuous Stirred Tank Reactor

The reaction dynamics for an isothermal continuous stirred tank reactor with a volume $V = 40$ liters and feed concentration $C_{A,f}$ are modeled as

$$\begin{aligned} V\frac{dC_A}{dt} &= q(C_{A,f} - C_A) - Vk_AC_A \\ V\frac{dC_B}{dt} &= -qC_B + Vk_AC_A - Vk_BC_B \end{aligned}$$

At steady-state the material balances become

$$\begin{aligned} 0 &= q(C_{A,f} - \bar{C}_A) - Vk_A\bar{C}_A \\ 0 &= -q\bar{C}_B + Vk_A\bar{C}_A - Vk_B\bar{C}_B \end{aligned}$$

which can be solved for C_A

$$\bar{C}_A = \frac{qC_{A,f}}{q + Vk_A}$$

and then for C_B

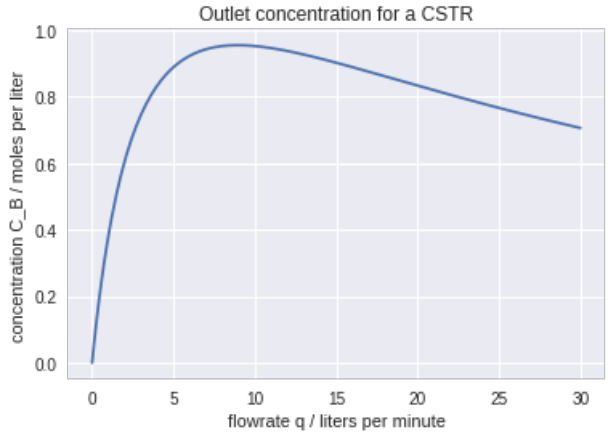
$$\bar{C}_B = \frac{qVk_AC_{A,f}}{(q + Vk_A)(q + Vk_B)}$$

The numerator is first-order in flowrate q, and the denominator is quadratic. This is consistent with an intermediate value of q corresponding to a maximum concentration $\bar{C}_B$.

The next cell plots $\bar{C}_B$ as a function of flowrate q.

In [9]:

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

V = 40      # liters
kA = 0.5    # 1/min
kB = 0.1    # 1/min
CAf = 2.0   # moles/liter

def cstr(q):
    return q*V*kA*CAf/(q + V*kB)/(q + V*kA)

q = np.linspace(0,30,200)
plt.plot(q, cstr(q))
plt.xlabel('flowrate q / liters per minute')
plt.ylabel('concentration C_B / moles per liter')
plt.title('Outlet concentration for a CSTR')
plt.show()
```



We see that, for the parameters given, there is an optimal flowrate somewhere between 5 and 10 liters per minute.

Analytical Solution using Calculus

As it happens, this problem has an interesting analytical solution that can be found by hand, and which can be used to check the accuracy of numerical solutions. Setting the first derivative of $\bar{C}_B$ to zero,

$$\left.\frac{d\bar{C}_B}{dq}\right|_{q^*} = \frac{Vk_AC_{A,f}}{(q^*+Vk_A)(q^*+Vk_B)} - \frac{q^*Vk_AC_{A,f}}{(q^*+Vk_A)^2(q^*+Vk_B)} - \frac{q^*Vk_AC_{A,f}}{(q^*+Vk_A)(q^*+Vk_B)^2} = 0$$

Clearing out the non-negative common factors yields

$$1 - \frac{q^*}{(q^*+Vk_A)} - \frac{q^*}{(q^*+Vk_B)} = 0$$

and multiplying by the non-negative denominators produces

$$q^{*2} + q^*V(k_A+k_B) + V^2k_Ak_B - q^*(q^*+Vk_B) - q^*(q^*+Vk_A) = 0$$

Expanding these expressions followed by arithmetic cancelations gives the final result

$$q^* = V\sqrt{k_Ak_B}$$

which shows the optimal dilution rate, $\frac{q^*}{V}$, is equal the geomtric mean of the rate constants.

In [10]:

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

V = 40      # liters
kA = 0.5    # 1/min
kB = 0.1    # 1/min
CAf = 2.0   # moles/liter

qmax = V*np.sqrt(kA*kB)
CBmax = cstr(qmax)
print('Flowrate at maximum CB = ', qmax, 'liters per minute.')
print('Maximum CB =', CBmax, 'moles per liter.')
print('Productivity = ', qmax*CBmax, 'moles per minute.')
```

```
Flowrate at maximum CB =  8.94427190999916 liters per minute.
Maximum CB = 0.9549150281252629 moles per liter.
Productivity =  8.541019662496845 moles per minute.
```

Numerical Solution with Pyomo

This problem can also be solved using Pyomo to create a model instance. First we make sure that Pyomo and ipopt are installed, then we proceed with the model specification and solution.

In [0]:

```
!pip install -q pyomo
!wget -N -q "https://ampl.com/dl/open/ipopt/ipopt-linux64.zip" && unzip -o -q ipopt-li
nux64
ipopt_executable = '/content/ipopt'
```

In [12]:

```
from pyomo.environ import *

V = 40      # liters
kA = 0.5    # 1/min
kB = 0.1    # 1/min
CAf = 2.0   # moles/liter

# create a model instance
m = ConcreteModel()

# create the decision variable
m.q = Var(domain=NonNegativeReals)

# create the objective
m.CBmax = Objective(expr=m.q*V*kA*CAf/(m.q + V*kB)/(m.q + V*kA), sense=maximize)

# solve using the nonlinear solver ipopt
SolverFactory('ipopt', executable=ipopt_executable).solve(m)

# print solution
print('Flowrate at maximum CB = ', m.q(), 'liters per minute.')
print('Maximum CB =', m.CBmax(), 'moles per liter.')
print('Productivity = ', m.q()*m.CBmax(), 'moles per minute.')
```

```
Flowrate at maximum CB =  8.944271964904415 liters per minute.
Maximum CB = 0.9549150281252627 moles per liter.
Productivity =  8.541019714926698 moles per minute.
```

One advantage of using Pyomo for solving problems like these is that you can reduce the amount of algebra needed to prepare the problem for numerical solution. This not only minimizes your work, but also reduces possible sources of error in your solution.

In this example, the steady-state equations are

$$\begin{aligned} 0 &= q(C_{A,f} - \bar{C}_A) - Vk_A\bar{C}_A \\ 0 &= -q\bar{C}_B + Vk_A\bar{C}_A - Vk_B\bar{C}_B \end{aligned}$$

with unknowns C_B and C_A. The modeling strategy is to introduce variables for the flowrate q and these unknowns, and introduce the steady state equations as constraints.

In [13]:

```
from pyomo.environ import *

V = 40     # liters
kA = 0.5   # 1/min
kB = 0.1   # 1/min
CAf = 2.0  # moles/liter

# create a model instance
m = ConcreteModel()

# create the decision variable
m.q  = Var(domain=NonNegativeReals)
m.CA = Var(domain=NonNegativeReals)
m.CB = Var(domain=NonNegativeReals)

# equations as constraints
m.eqn = ConstraintList()
m.eqn.add(0 == m.q*(CAf - m.CA) - V*kA*m.CA)
m.eqn.add(0 == -m.q*m.CB + V*kA*m.CA - V*kB*m.CB)

# create the objective
m.CBmax = Objective(expr=m.CB, sense=maximize)

# solve using the nonlinear solver ipopt
SolverFactory('ipopt', executable=ipopt_executable).solve(m)

# print solution
print('Flowrate at maximum CB = ', m.q(), 'liters per minute.')
print('Maximum CB =', m.CBmax(), 'moles per liter.')
print('Productivity = ', m.q()*m.CBmax(), 'moles per minute.')
```

```
Flowrate at maximum CB =  8.944272002876573 liters per minute.
Maximum CB = 0.9549150281377385 moles per liter.
Productivity =  8.541019751298471 moles per minute.
```

In [0]:

```

```

6.2 Maximizing Concentration of an Intermediate in a Batch Reactor

In [0]:

```
!pip install -q pyomo
!wget -N -q "https://ampl.com/dl/open/ipopt/ipopt-linux64.zip"
!unzip -o -q ipopt-linux64

ipopt_executable = '/content/ipopt'
```

Mathematical Model

A material balance for an isothermal stirred batch reactor with a volume $V = 40$ liters and an initial concentration $C_{A,f}$ is given by

$$\begin{aligned} V\frac{dC_A}{dt} &= -Vk_AC_A \\ V\frac{dC_B}{dt} &= Vk_AC_A - Vk_BC_B \end{aligned}$$

Eliminating the common factor V

$$\begin{aligned} \frac{dC_A}{dt} &= -k_AC_A \\ \frac{dC_B}{dt} &= k_AC_A - k_BC_B \end{aligned}$$

With an initial concentration $C_{A,f}$. A numerical solution to these equations is shown in the following cell.

In [3]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

V = 40      # liters
kA = 0.5    # 1/min
kB = 0.1    # 1/min
CAf = 2.0   # moles/liter

def batch(X, t):
    CA, CB = X
    dCA_dt = -kA*CA
    dCB_dt = kA*CA - kB*CB
    return [dCA_dt, dCB_dt]

t = np.linspace(0,30,200)
soln = odeint(batch, [CAf,0], t)
plt.plot(t, soln)
plt.xlabel('time / minutes')
plt.ylabel('concentration / moles per liter')
plt.title('Batch Reactor')
plt.legend(['$C_A$','$C_B$'])
```

Out[3]:

```
<matplotlib.legend.Legend at 0x7f57786aceb8>
```

Optimization with `scipy.minimize_scalar`

To find the maximum value, we first write a function to compute C_B for any value of time t.

In [0]:

```
def CB(tf):
    soln = odeint(batch, [CAf, 0], [0, tf])
    return soln[-1][1]
```

We gain use `minimize_scalar` to find the value of t that minimizes the negative value of $C_B(t)$.|

In [5]:

```
from scipy.optimize import minimize_scalar
minimize_scalar(lambda t: -CB(t), bracket=[0,50])
```

Out[5]:

```
     fun: -1.3374806339222158
    nfev: 23
     nit: 19
 success: True
       x: 4.023594924340666
```

In [6]:

```
tmax = minimize_scalar(lambda t: -CB(t), bracket=[0,50]).x

print('Concentration c_B has maximum', CB(tmax), 'moles/liter at time', tmax,
'minutes.')
```

```
Concentration c_B has maximum 1.3374806339222158 moles/liter at time 4.023594924340666
minutes.
```

Solution Using Pyomo

The variable to be found is the time t_f corresponding to the maximum concentration of B. For this purpose we introduce a scaled time

$$\tau = \frac{t}{t_f}$$

so that $\tau = 1$ as the desired solution. The problem then reads

$$\max_{t_f} C_B(\tau = 1)$$

subject to

$$\begin{aligned} \frac{dC_A}{d\tau} &= -t_f k_A C_A \\ \frac{dC_B}{d\tau} &= t_f(k_A C_A - k_B C_B) \end{aligned}$$

The solution to this problem is implemented as a solution to the following Pyomo model.

In [8]:

```
from pyomo.environ import *
from pyomo.dae import *

V   = 40    # liters
kA  = 0.5   # 1/min
kB  = 0.1   # 1/min
cAf = 2.0   # moles/liter

m = ConcreteModel()

m.tau = ContinuousSet(bounds=(0, 1))

m.tf = Var(domain=NonNegativeReals)
m.cA = Var(m.tau, domain=NonNegativeReals)
m.cB = Var(m.tau, domain=NonNegativeReals)

m.dcA = DerivativeVar(m.cA)
m.dcB = DerivativeVar(m.cB)

m.odeA = Constraint(m.tau,
    rule=lambda m, tau: m.dcA[tau] == m.tf*(-kA*m.cA[tau]) if tau > 0 else Constraint.Sk
ip)
m.odeB = Constraint(m.tau,
    rule=lambda m, tau: m.dcB[tau] == m.tf*(kA*m.cA[tau] - kB*m.cB[tau]) if tau > 0 else
Constraint.Skip)

m.ic = ConstraintList()
m.ic.add(m.cA[0]  == cAf)
m.ic.add(m.cB[0]  == 0)

m.obj = Objective(expr=m.cB[1], sense=maximize)

TransformationFactory('dae.collocation').apply_to(m)
SolverFactory('ipopt', executable=ipopt_executable).solve(m)
print('Concentration c_B has maximum', m.cB[1](), 'moles/liter at time', m.tf(), 'minut
es.')
```

```
Concentration c_B has maximum 1.3374805810221073 moles/liter at time 4.023594178375687
minutes.
```

In [0]:

```
```

6.3 Path Planning for a Simple Car

In [ ]:

```
!pip install -q pyomo
!wget -N -q "https://ampl.com/dl/open/ipopt/ipopt-linux64.zip"
!unzip -o -q ipopt-linux64

ipopt_executable = '/content/ipopt'
```

Pyomo Model

In [2]:

```
from pyomo.environ import *
from pyomo.dae import *

L = 2
tf = 50

# create a model object
m = ConcreteModel()

# define the independent variable
m.time = ContinuousSet(bounds=(0, tf))

# define control inputs
m.a = Var(m.time)
m.v = Var(m.time, domain=Reals, bounds=(-0.1,0.1))

# define the dependent variables
m.x = Var(m.time)
m.y = Var(m.time)
m.t = Var(m.time)
m.u = Var(m.time)
m.p = Var(m.time, domain=Reals, bounds=(-0.5,0.5))

m.dxdt = DerivativeVar(m.x)
m.dydt = DerivativeVar(m.y)
m.dtdt = DerivativeVar(m.t)
m.dudt = DerivativeVar(m.u)
m.dpdt = DerivativeVar(m.p)

# define the differential equation as a constraint
m.ode_x = Constraint(m.time, rule=lambda m, time: m.dxdt[time] ==
m.u[time]*cos(m.t[time]))
m.ode_y = Constraint(m.time, rule=lambda m, time: m.dydt[time] ==
m.u[time]*sin(m.t[time]))
m.ode_t = Constraint(m.time, rule=lambda m, time: m.dtdt[time] ==
m.u[time]*tan(m.p[time])/L)
m.ode_u = Constraint(m.time, rule=lambda m, time: m.dudt[time] == m.a[time])
m.ode_p = Constraint(m.time, rule=lambda m, time: m.dpdt[time] == m.v[time])

# path constraints
m.path_x1 = Constraint(m.time, rule=lambda m, time: m.x[time] >= 0)
m.path_y1 = Constraint(m.time, rule=lambda m, time: m.y[time] >= 0)

# initial conditions
m.ic = ConstraintList()
m.ic.add(m.x[0]==0)
m.ic.add(m.y[0]==0)
m.ic.add(m.t[0]==0)
```

```
m.ic.add(m.u[0]==0)
m.ic.add(m.p[0]==0)

# final conditions
m.fc = ConstraintList()
m.fc.add(m.x[tf]==0)
m.fc.add(m.y[tf]==20)
m.fc.add(m.t[tf]==0)
m.fc.add(m.u[tf]==0)
m.fc.add(m.p[tf]==0)

# define the optimization objective
m.integral = Integral(m.time, wrt=m.time,
                      rule=lambda m, time: 0.2*m.p[time]**2 + m.a[time]**2 + m.v[time]**
2)
m.obj = Objective(expr=m.integral)

# transform and solve
TransformationFactory('dae.collocation').apply_to(m, wrt=m.time, nfe=3, ncp=12, method=
'BACKWARD')
solver = SolverFactory('ipopt', executable=ipopt_executable)
solver.solve(m).write()
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Lower bound: -inf
  Upper bound: inf
  Number of objectives: 1
  Number of constraints: 449
  Number of variables: 444
  Sense: unknown
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Message: Ipopt 3.12.8\x3a Optimal Solution Found
  Termination condition: optimal
  Id: 0
  Error rc: 0
  Time: 1.157731294631958
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

Accessing Solution Data

In [ ]:

```
# access the results
time = [time for time in m.time]

a = [m.a[time]() for time in m.time]
v = [m.v[time]() for time in m.time]

x = [m.x[time]() for time in m.time]
y = [m.y[time]() for time in m.time]
t = [m.t[time]() for time in m.time]
u = [m.u[time]() for time in m.time]
p = [m.p[time]() for time in m.time]
```

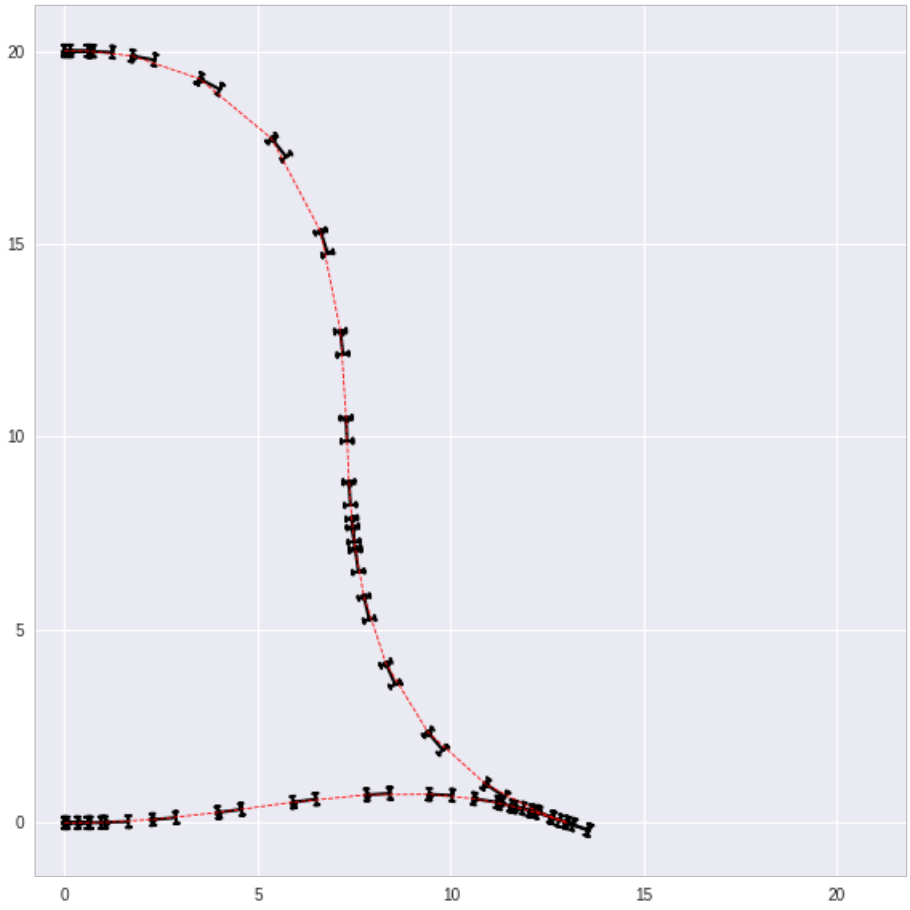
Visualizing the Car Path

In [4]:

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

scl=0.3

def draw_car(x=0, y=0, theta=0, phi=0):
    R = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])
    car = np.array([[0.2, 0.5], [-0.2, 0.5], [0, 0.5], [0, -0.5],
                    [0.2, -0.5], [-0.2, -0.5], [0, -0.5], [0, 0], [L, 0], [L, 0.5],
                    [L + 0.2*np.cos(phi), 0.5 + 0.2*np.sin(phi)],
                    [L - 0.2*np.cos(phi), 0.5 - 0.2*np.sin(phi)], [L, 0.5],[L, -0.5],
                    [L + 0.2*np.cos(phi), -0.5 + 0.2*np.sin(phi)],
                    [L - 0.2*np.cos(phi), -0.5 - 0.2*np.sin(phi)]])
    carz= scl*R.dot(car.T)
    plt.plot(x + carz[0], y + carz[1], 'k', lw=2)
    plt.plot(x, y, 'k.', ms=10)

plt.figure(figsize=(10,10))
for xs,ys,ts,ps in zip(x,y,t,p):
    draw_car(xs, ys, ts, scl*ps)
plt.plot(x, y, 'r--', lw=0.8)
plt.axis('square')
```

Out[4]:

```
(-0.7475794562067691,
 21.812257339073334,
 -1.3839156648530575,
 21.175921130427046)
```

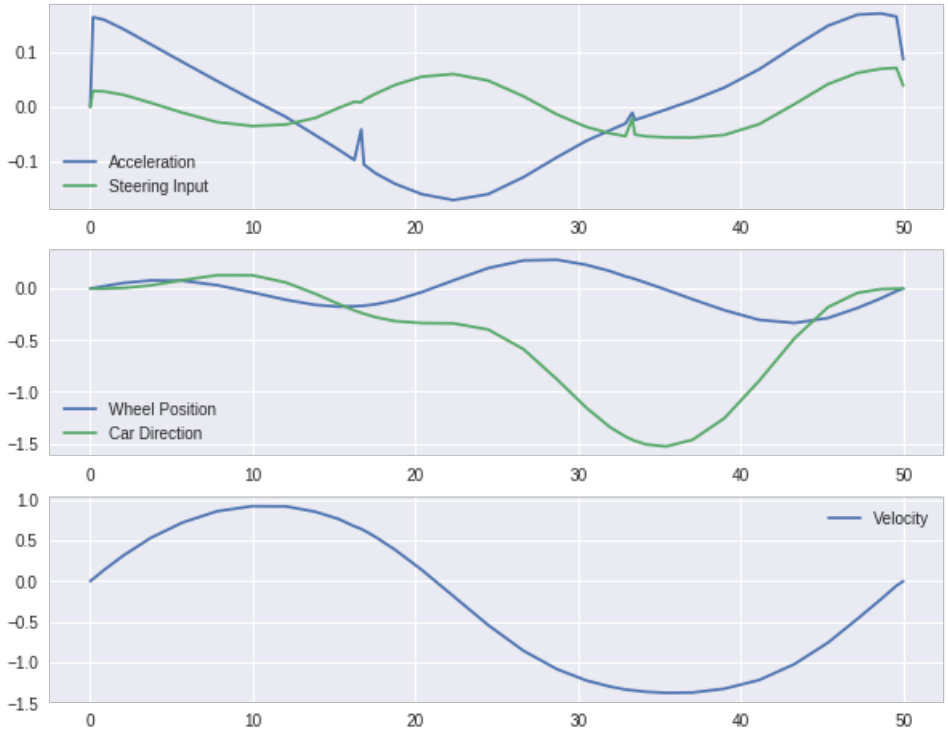


In [5]:

```
plt.figure(figsize=(10,8))
plt.subplot(311)
plt.plot(time, a, time, v)
plt.legend(['Acceleration','Steering Input'])

plt.subplot(312)
plt.plot(time, p, time, t)
plt.legend(['Wheel Position','Car Direction'])

plt.subplot(313)
plt.plot(time, u)
plt.legend(['Velocity'])
```

Out[5]:

```
<matplotlib.legend.Legend at 0x7ff2c45977b8>
```



In [ ]:

Chapter 7. Parameter Estimation

7.1 Parameter Estimation

In [ ]:

```
!pip install -q pyomo
!wget -N -q "https://ampl.com/dl/open/ipopt/ipopt-linux64.zip"
!unzip -o -q ipopt-linux64

ipopt_executable='/content/ipopt'
```

In [ ]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Steps in Model Fitting

The goal of parameter estimation is to use experimental data to estimate values of the unknown parameters in model describing the system of interest, and to quantify uncertainty associated with those estimates.

Following the outline in "Model Fitting and Error Estimation", Costa, Kleinstein, Hershberg :

1. Assemble the data in useful data structures. Plot the data. Be sure there is meaningful variations to enable parameter estimation.
2. Select an appropriate model based on underlying theory. Identify the known and unknown parameters.
3. Define a "figure of merit" function the measures the agreement of model to data for given value of the unknown parameters.
4. Adjust the unknown parameters to find the best fit. Generally this involves minimizing the figure of merit function.
5. Evaluate the "Goodness of Fit". Test assumptions of randomness, and for systematic deviations between the model and data.
6. Estimate the accuracy of the best-fit parameters. Provide confidence intervals and covariance among unknown parameters.
7. Determine whether a better fit is possible.

This notebook uses the Pandas package to encapuslate and display experimental data. Pandas is a comprehensive Python package for managing and displaying data. It provides many useful and time-saving tools for performing common operations on data.

Step 1. The Data

The following data is taken from ["Lecture Notes for CBE 20258" by Prof. David Leighton]. The data consists of measurements done for three different feed concentrations `C0`. For each feed concentration, students adjust reactor temperature `T` using an external heat source, then measure the reactor exit concentration `C`.

For convenience, the data is first encoded as a nested Python dictionary. The first index refers to the name of the experiment. The data for each experiment is then encoded as a nested dictionary with name, value pairs. The values are strings, numbers, or lists of measured responses. The name of the experiment is repeated to facilitate conversion to a Pandas data frame.

In [ ]:

```
data = {
    'Expt A': {
        'Expt': 'A',
        'C0'  : 1.64E-4,
        'T'   : [423, 449, 471, 495, 518, 534, 549, 563],
        'C'   : [1.66e-4, 1.66e-4, 1.59e-4, 1.37e-4, 8.90e-5, 5.63e-5, 3.04e-5, 1.71e-5
,
    },
    'Expt B': {
        'Expt': 'B',
        'C0'  : 3.69e-4 ,
        'T'   : [423, 446, 469, 490, 507, 523, 539, 553, 575],
        'C'   : [3.73e-4, 3.72e-4, 3.59e-4, 3.26e-4, 2.79e-4, 2.06e-4, 1.27e-4, 7.56e-5
3.76e-5],
    },
    'Expt C': {
        'Expt': 'C',
        'C0'  : 2.87e-4,
        'T'   : [443, 454, 463, 475, 485, 497, 509, 520, 534, 545, 555, 568],
        'C'   : [2.85e-4, 2.84e-4, 2.84e-4, 2.74e-4, 2.57e-4, 2.38e-4, 2.04e-4, 1.60e-4
1.12e-4,
                 6.37e-5, 5.07e-5, 4.49e-5],
    },
}
```

The following cell converts the data dictionary of experimental data to a Pandas dataframe then prints the result.

In [4]:

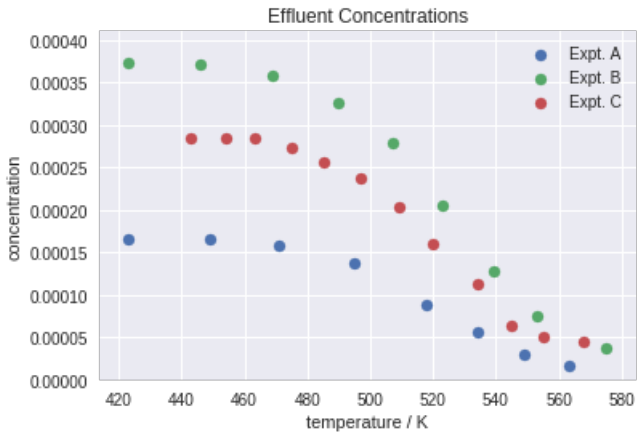
```
df = pd.concat([pd.DataFrame(data[expt]) for expt in data])
df = df.set_index('Expt')
print(df)
```

```
              C         C0    T
Expt
A      0.000166  0.000164  423
A      0.000166  0.000164  449
A      0.000159  0.000164  471
A      0.000137  0.000164  495
A      0.000089  0.000164  518
A      0.000056  0.000164  534
A      0.000030  0.000164  549
A      0.000017  0.000164  563
B      0.000373  0.000369  423
B      0.000372  0.000369  446
B      0.000359  0.000369  469
B      0.000326  0.000369  490
B      0.000279  0.000369  507
B      0.000206  0.000369  523
B      0.000127  0.000369  539
B      0.000076  0.000369  553
B      0.000038  0.000369  575
C      0.000285  0.000287  443
C      0.000284  0.000287  454
C      0.000284  0.000287  463
C      0.000274  0.000287  475
C      0.000257  0.000287  485
C      0.000238  0.000287  497
C      0.000204  0.000287  509
C      0.000160  0.000287  520
C      0.000112  0.000287  534
C      0.000064  0.000287  545
C      0.000051  0.000287  555
C      0.000045  0.000287  568
```

It's generally a good idea to create some initial displays of the experimental data before attempting to fit a mathematical model. This provides an opportunity to identify potential problems with data, observe key features, and look for particular symmetries or patterns in the data.

In [5]:

```
for expt in sorted(set(df.index)):
    plt.scatter(df['T'][expt], df['C'][expt])

plt.ylim(0, 1.1*max(df['C']))
plt.xlabel('temperature / K')
plt.ylabel('concentration')
plt.legend(["Expt. " + expt for expt in sorted(set(df.index))])
plt.title('Effluent Concentrations');
```



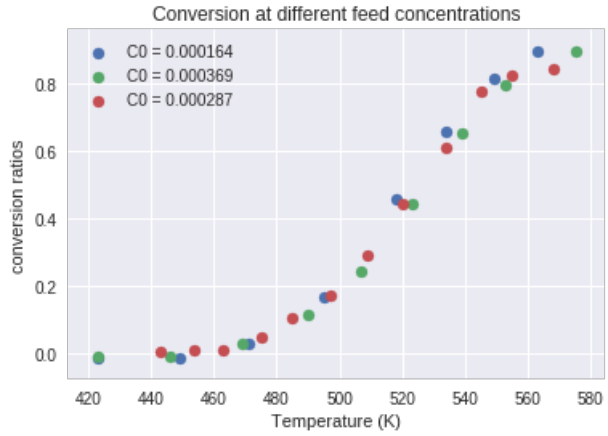
An initial plot of the data shows the effluent concentration decreases at higher operating temperatures. This indicates higher conversion at higher temperatures. A plot of conversion `X`

$$X = \frac{C_0 - C}{C_0}$$

collapses the data into a what appears to be a single function of temperature. The following cell adds conversion as an additional column of the data set.

In [6]:

```
# add a column 'X' to the dataframe
df['X'] = 1 - df['C']/df['C0']

for expt in sorted(set(df.index)):
    plt.scatter(df['T'][expt], df['X'][expt])

plt.xlabel('Temperature (K)')
plt.ylabel('conversion ratios')
plt.legend(['C0 = ' + str(list(df['C0'][expt])[0]) for expt in sorted(set(df.index))])
plt.title('Conversion at different feed concentrations');
```



Step 2. Select a Model

A proposed steady-state model for the catalytic reactor is

$$0 = (C_0 - C) - \frac{m}{q} k_0 C^n \left(\frac{T}{T_r}\right)^n e^{-\frac{E_a}{RT_r}\frac{T_r}{T}}$$

The known parameters are the flow rate q, the mass of catalysit m, and a reference temperature T_r. The ratio $\frac{T}{T_r}$ is the 'reduced temperature' and provides for a better conditioned equation. The unknown parameters are the reaction order n, the Arrhenius pre-exponental factor k_0, and the reduced activation energy $\frac{E_a}{RT_r}$. Generally the pre-exponential factor is very large, therefore is combined within the exponential term as $\ln k_0$.

$$0 = (C_0 - C) - \frac{m}{q} C^n \left(\frac{T}{T_r}\right)^n e^{\ln k_0 - \frac{E_a}{RT_r}\frac{T_r}{T}}$$

The unknown parameters are:

Parameter	Code	Description
n	`n`	reaction order
$\ln k_0$	`lnk0`	natural log of Arrenhius pre-exponential factor
$\frac{E_a}{RT_r}$	`ERTr`	reduced activation energy

Step 3. Define a "figure of merit"

Each experimental measure consists of values of the unknown parameters and experimental values $C_{0,k}$, T_k, and C_k. Given estimates for the unknown parameters, the model equation provides a convenient definition of a residual r_k as

$$r_k = C_{0,k} - C_k - \frac{m}{q} C^n \left(\frac{T_k}{T_r} \right)^n e^{\ln k_0 - \frac{E_a}{RT_r}\frac{T_r}{T_k}}$$

If the model is an accurate depiction of the reaction processes then we expect the residuals to be small, random variates.

In [ ]:

```
Tr = 298     # reference temperature.
q = 0.1      # flow rate (liters/min)
m = 1        # amount of catalyst (g)

def residuals(parameters, df):
    n, lnk0, ERTr = parameters
    C0, C, T = df['C0'], df['C'], df['T']
    return C0 - C - (m/q) * C**n  * (T/Tr)**n * np.exp(lnk0 - ERTr*Tr/T)
```

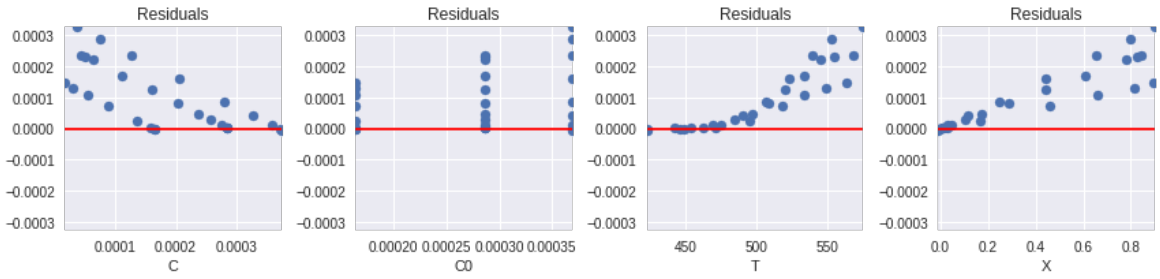
To illustrate, the following cell calculates the residuals for an initial estimate for the unknown parameter values. The residuals are then plotted as a function of the experimental variables to see if the residuals behave as random variates.

In [8]:

```
parameter_names = ['n', 'lnk0', 'ERTr']
parameter_guess = [1, 15, 38]

def plot_residuals(r, df, ax=None):
    rmax = np.max(np.abs(r))
    if ax is None:
        fig, ax = plt.subplots(1, len(df.columns), figsize=(12,3))
    else:
        rmax = max(ax[0].get_ylim()[1], rmax)
    n = 0
    for c in df.columns:
        ax[n].scatter(df[c], r)
        ax[n].set_ylim(-rmax, rmax)
        ax[n].set_xlim(min(df[c]), max(df[c]))
        ax[n].plot(ax[n].get_xlim(), [0,0], 'r')
        ax[n].set_xlabel(c)
        ax[n].set_title('Residuals')
        n += 1
    plt.tight_layout()
    return ax

r = residuals(parameter_guess, df)
plot_residuals(r, df)
```

Out[8]:

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fcfe2535668>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fcfe2456048>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fcfe2488b38>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fcfe2446208>],
      dtype=object)
```



Step 4. Find a Best Fit

A least squares 'figure of merit' for the fit of the model to the experimental data is given by

$$SOS = \sum_k r_k^2$$

Our goal is to find values for the unknown parameters that minimize the sum of the squares of the residuals.

The following cell defines two functions. The first is `sos` which calculates the sum of squares of the residuals. The `best_fit` function uses `scipy.optimize.fmin`

In [9]:

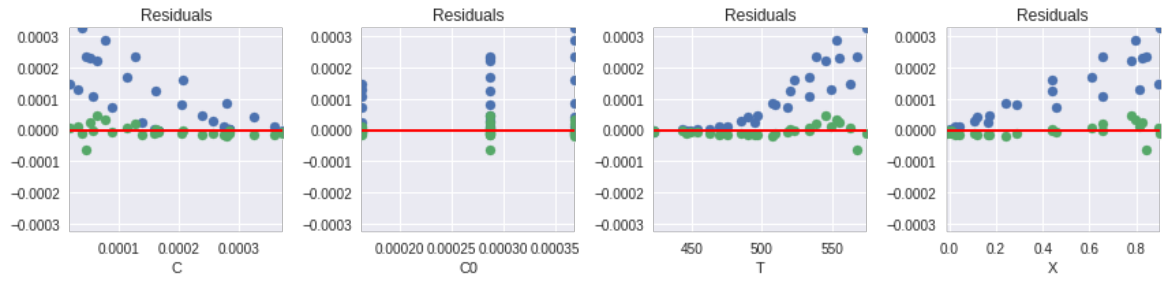
```
from scipy.optimize import fmin

def sos(parameters, df):
    return sum(r**2 for r in residuals(parameters, df))

def best_fit(fcn, df, disp=1):
    return fmin(fcn, parameter_guess, args=(df,), disp=disp)

parameter_fit = best_fit(sos, df)

for name,value in zip(parameter_names, parameter_fit):
    print(name, " = ", round(value,2))
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 140
         Function evaluations: 257
n   =  0.65
lnk0   =  13.65
ERTr   =  34.21
```

Let's compare how the residuals have been reduced as a result of minimizing the sum of squares.

In [10]:

```
ax = plot_residuals(residuals(parameter_guess, df), df)
plot_residuals(residuals(parameter_fit, df), df, ax=ax);
```



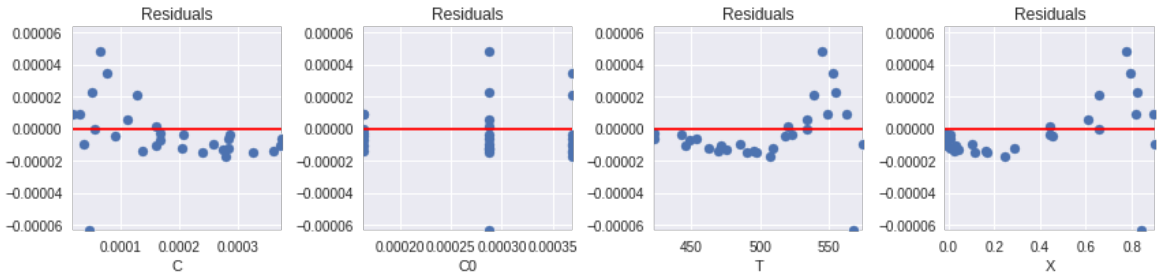
Step 5. Evaluate the Goodness of Fit

Plotting Residuals

An important element of any parameter fitting exercise is to examine the residuals for systematic errors. The following cell plots the residuals as functions of .

In [11]:

```
r = residuals(parameter_fit, df)
plot_residuals(r, df);
```



It's apparent that there is systematic error in the residuals. To cause the minimizer to put more weight on those large residuals, a scaling factor is introduced into the norm used to measure the residual for the purpose of parameter estimation.

Separable Model

An interesting feature of the model for the catalytic reactor is that it can be separated into two sides such that the right-hand side is a function only of T.

$$\frac{C_0 - C}{C^n} = \frac{m}{q}\left(\frac{T}{T_r}\right)^n e^{\ln k_0 - \frac{E_a}{RT_r}\frac{T_r}{T}}$$

Note that the left-hand side is a function of two independent experimental variables. But if the model accurately represents experimental behavior, then a plot of the left-hand side versus temperature should collapse to a single curve.

The next cell plots the experimental data as pairs (x_k, y_k) where

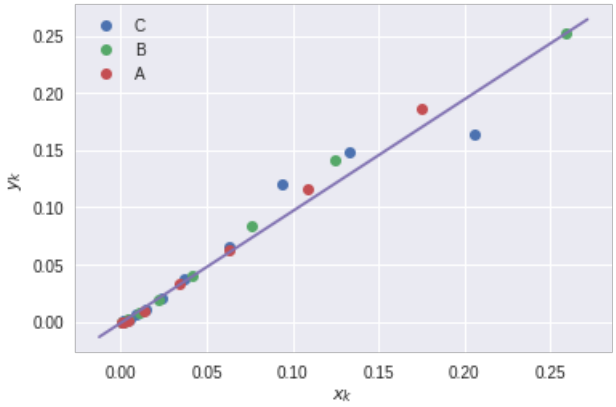
$$x_k = \frac{C_{0,k} - C_k}{C_k^n}$$

$$y_k = \frac{m}{q}\left(\frac{T_k}{T_r}\right)^n e^{\ln k_0 - \frac{E_a}{RT_r}\frac{T_r}{T_k}}$$

If the fitted model is an accurate representation of the catalytic reactor, then the data should collapse to single curve aligned with diagonal.

In [12]:

```
n, lnk0, ERTr = parameter_fit
for expt in set(df.index):
    y = (df['C0'][expt] - df['C'][expt])/df['C'][expt]**n
    x = (m/q) * (df['T'][expt]/Tr)**n * np.exp(lnk0 - ERTr*Tr/df['T'][expt])
    plt.plot(x, y, marker='o', lw=0)
plt.plot(plt.xlim(), plt.ylim())
plt.ylabel('$y_k$')
plt.xlabel('$x_k$')
plt.legend(set(df.index));
```



Substituting $C = C_0(1 - X)$

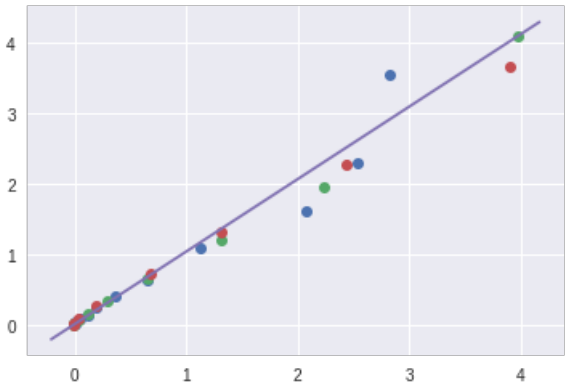
$$\frac{X}{(1-X)^n}\frac{1}{C_0^{n-1}} = \frac{m}{q}\left(\frac{T}{T_r}\right)^n e^{\ln k_0 - \frac{E_a}{RT_r}\frac{T_r}{T}}$$

In [13]:

```
for expt in set(df.index):
    y = df['C0'][expt]**(n-1) * (m/q) * (df['T'][expt]/Tr)**n * np.exp(lnk0 - ERTr*Tr/df
['T'][expt])
    x = df['X'][expt]/(1 - df['X'][expt])**n
    plt.plot(x, y, marker='o', lw=0)
plt.plot(plt.xlim(), plt.ylim())
plt.xlabel('')
```

Out[13]:

```
Text(0.5,0,'')
```



Step 6. Estimate Confidence Intervals

The following cell implements a simple bootstrap method for estimating confidence intervals for the fitted parameters. The function `resample` creates a new dataframe of data by sampling (with replacement) the original experimental data.

The `best_fit` function is called `N` times on resampled data sets to create a statistical sample of fitted parameters. Those results are analyzed using statistical functions from the Pandas library.

In [ ]:

```
from random import choices

def resample(df):
    return df.iloc[choices(range(0, len(df)), k=len(df))]

N = 100
dp = pd.DataFrame([best_fit(sos, resample(df), disp=0) for k in range(0,N)], columns=pa
rameter_names)
```

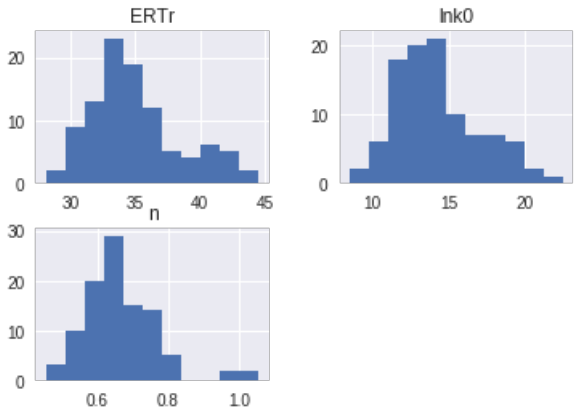
In [15]:

```
print('Mean values and 95% confidence interval')
for p in dp.columns:
    print(p, " = ", round(dp[p].mean(),2), " +/- ", round(1.96*dp[p].std(),2))

dp.hist(bins=1+int(np.sqrt(N)))
print("\n\nQuantiles")
dp.quantile([0.05, .5, 0.95]).T
```

```
Mean values and 95% confidence interval
n  =  0.67  +/-  0.2
lnk0  =  14.23  +/-  5.34
ERTr  =  35.02  +/-  6.86


Quantiles
```

Out[15]:

	0.05	0.5	0.95
n	0.528570	0.652235	0.825784
lnk0	10.370035	13.788439	18.946778
ERTr	30.430791	34.260671	41.794342



Step 7. Determine if a Better Fit is Possible

In this section we introduce an alternative method of coding solutions to parameter estimation problems using Pyomo. An advantage of using Pyomo is the more explicit indentification of unknown parameters, objective, and constraints on parameter values.

Pyomo Model - Version 1

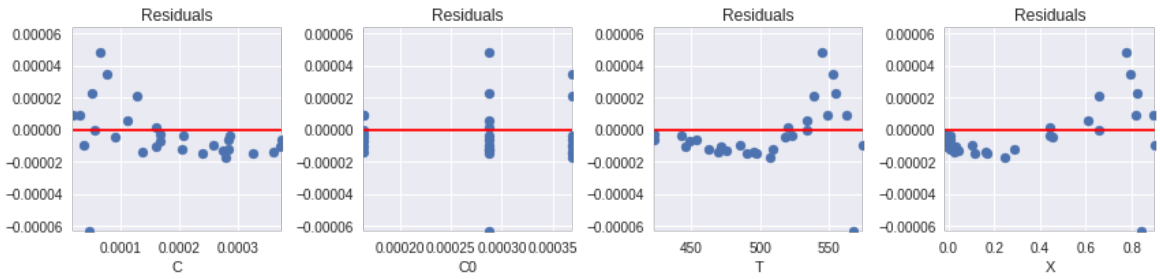
The first version of a Pyomo model for estimating parameters for the catalytic reactor is a direct translation of the approach outlined above. There are some key coding considerations in performing this translation to a Pyomo model:

- As of the current version of Pyomo, an optimization variable cannot appear in the exponent of an expresson. The workaound is to recognize
$$C^n = e^{n \ln C}$$
- The residuals are expressed as a list of Pyomo expressions.

In [16]:

```
from pyomo.environ import *

Tr = 298    # The reference temperature.
q = 0.1     # The flow rate (liters/min)
m = 1       # The amount of catalyst (g)

def pyomo_fit1(df):
    T  = list(df['T'])
    C  = list(df['C'])
    C0 = list(df['C0'])

    mdl = ConcreteModel()
    mdl.n    = Var(domain=Reals, initialize=1)
    mdl.lnk0 = Var(domain=Reals, initialize=15)
    mdl.ERTr = Var(domain=Reals, initialize=38)

    residuals = [
        C0[k] - C[k] - (m/q) * exp(mdl.n*log(C[k])) * exp(mdl.n*log(T[k]/Tr)) * exp(mdl.l
nk0 - mdl.ERTr*Tr/T[k])
        for k in range(len(C))]

    mdl.obj = Objective(expr=sum(residuals[k]**2 for k in range(len(C))),
sense=minimize)
    SolverFactory('ipopt', executable=ipopt_executable).solve(mdl)
    return [mdl.n(), mdl.lnk0(), mdl.ERTr()], [residuals[k]() for k in range(len(C))]

parameter_values_1, r1 = pyomo_fit1(df)
plot_residuals(r1, df)
for name,value in zip(parameter_names, parameter_values_1):
    print(name, " = ", round(value,2))
```

```
n   =  0.65
lnk0   =  13.65
ERTr   =  34.21
```



Pyomo Model - Version 2

As a second model we consider a somewhat simpler model that omits the temperature dependence of the Arrhenius pre-exponential factor:

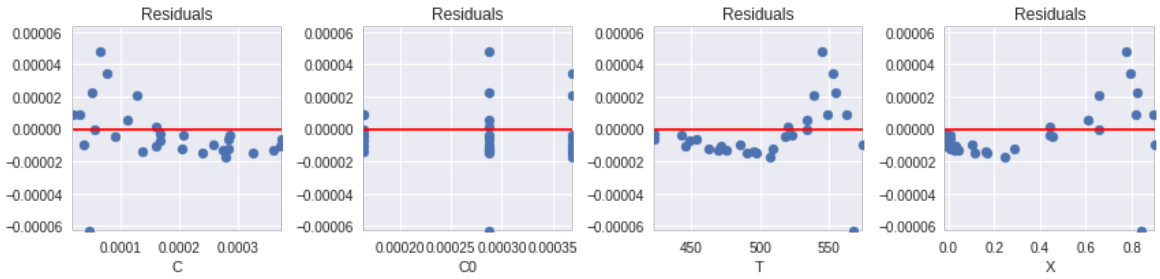
$$0 = C_0 - C - \frac{m}{q} C^n e^{\ln k_0 - \frac{E_a}{RT_r}\frac{T_r}{T}}$$

Can this somewhat simpler model adequately fit the data?

In [17]:

```
from pyomo.environ import *

Tr = 298    # The reference temperature.
q = 0.1     # The flow rate (liters/min)
m = 1       # The amount of catalyst (g)

def pyomo_fit2(df):
    T  = list(df['T'])
    C  = list(df['C'])
    C0 = list(df['C0'])

    mdl = ConcreteModel()
    mdl.n    = Var(domain=Reals, initialize=1)
    mdl.lnk0 = Var(domain=Reals, initialize=15)
    mdl.ERTr = Var(domain=Reals, initialize=38)

    residuals = [
        C0[k] - C[k] - (m/q) * exp(mdl.n*log(C[k])) * exp(mdl.lnk0 - mdl.ERTr*Tr/T[k])
        for k in range(len(C))]

    mdl.obj = Objective(expr=sum(residuals[k]**2 for k in range(len(C))),
sense=minimize)
    SolverFactory('ipopt', executable=ipopt_executable).solve(mdl)
    return [mdl.n(), mdl.lnk0(), mdl.ERTr()], [residuals[k]() for k in range(len(C))]

parameter_values_2, r2 = pyomo_fit2(df)
plot_residuals(r2, df)
for name,value in zip(parameter_names, parameter_values_2):
    print(name, " = ", round(value,2))
```

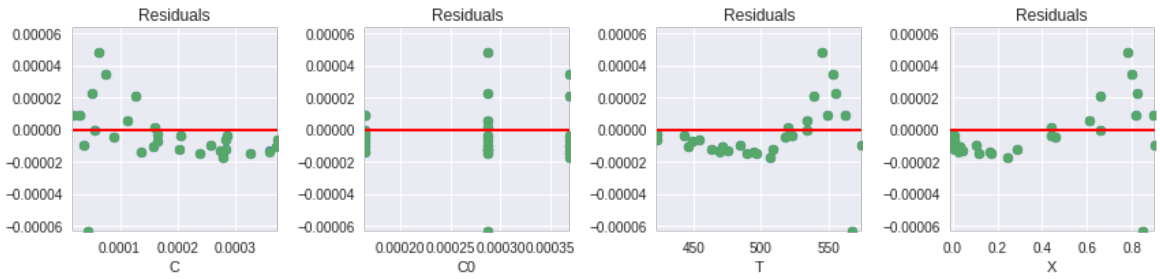
```
n   =  0.65
lnk0   =  14.69
ERTr   =  35.4
```



Is there a meaningful difference between these two models?

In [18]:

```
ax = plot_residuals(r2, df)
plot_residuals(r1, df, ax=ax);
```

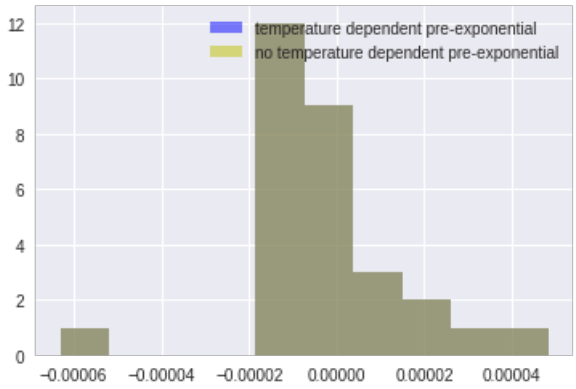


In [19]:

```
plt.hist(r1, color='b', alpha=0.5)
plt.hist(r2, color='y', alpha=0.5)
plt.legend(['temperature dependent pre-exponential', 'no temperature dependent pre-exponential'])
```

Out[19]:

```
<matplotlib.legend.Legend at 0x7fcfe0114ef0>
```



Based on this analysis of residuals, there is very little statistical support for a temperature dependent Arrhenius pre-exponential factor. Generally speaking, in situations like this it is generally wise to go with the simplist (i.e, most parsimonious) model that can adequately explain the data.

Pyomo Model - Version 3

Is the reaction n^{th} order, or could it be satisfactorily approximated with first-order kinetics?

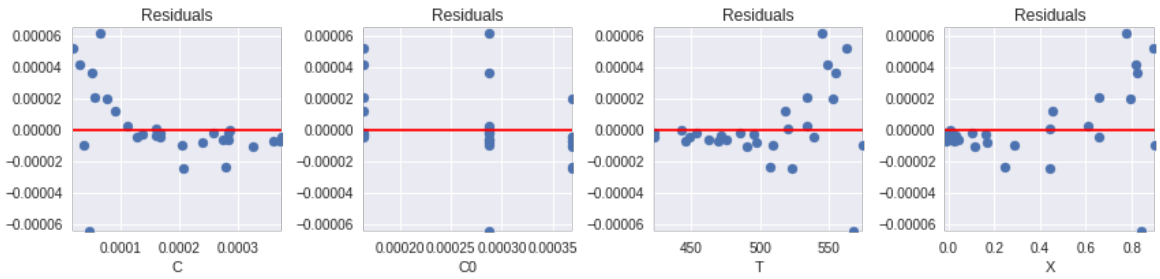
$$0 = C_0 - C - \frac{m}{q} C e^{\ln k_0 - \frac{E_a}{RT_r}\frac{T_r}{T}}$$

The following cell adds a constraint to force $n = 1$. We'll then examine the residuals to test if there is a statistically meaningful difference between the case $n = 1$ and $n \neq 1$.

In [20]:

```
from pyomo.environ import *

Tr = 298    # The reference temperature.
q = 0.1     # The flow rate (liters/min)
m = 1       # The amount of catalyst (g)

def pyomo_fit3(df):
    T  = list(df['T'])
    C  = list(df['C'])
    C0 = list(df['C0'])

    mdl = ConcreteModel()
    mdl.n    = Var(domain=Reals, initialize=1)
    mdl.lnk0 = Var(domain=Reals, initialize=15)
    mdl.ERTr = Var(domain=Reals, initialize=38)

    residuals = [
        C0[k] - C[k] - (m/q) * exp(mdl.n*log(C[k])) * exp(mdl.lnk0 - mdl.ERTr*Tr/T[k])
        for k in range(len(C))]

    mdl.obj = Objective(expr=sum(residuals[k]**2 for k in range(len(C))),
sense=minimize)
    mdl.con = Constraint(expr = mdl.n==1)
    SolverFactory('ipopt', executable=ipopt_executable).solve(mdl)
    return [mdl.n(), mdl.lnk0(), mdl.ERTr()], [residuals[k]() for k in range(len(C))]

parameter_values_3, r3 = pyomo_fit3(df)
plot_residuals(r3, df)
for name,value in zip(parameter_names, parameter_values_3):
    print(name, " = ", round(value, 2))
```

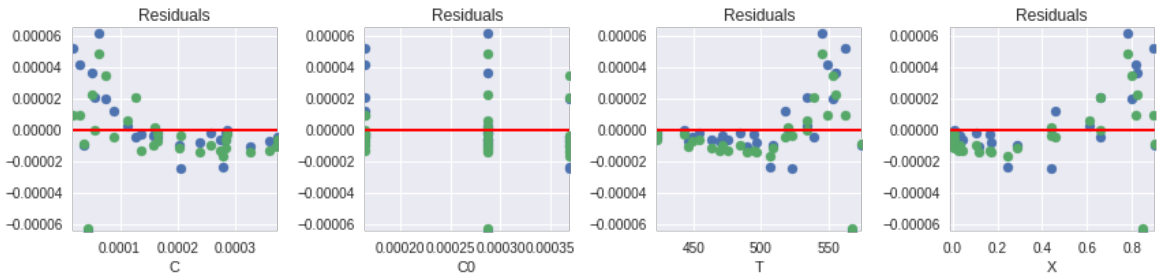
```
n   =  1.0
lnk0   =  23.01
ERTr   =  44.59
```



Comparing to models above, we see that restricting $n = 1$ does lead to somewhat larger residuals.

In [21]:

```
ax = plot_residuals(r3, df)
plot_residuals(r2, df, ax=ax);
```

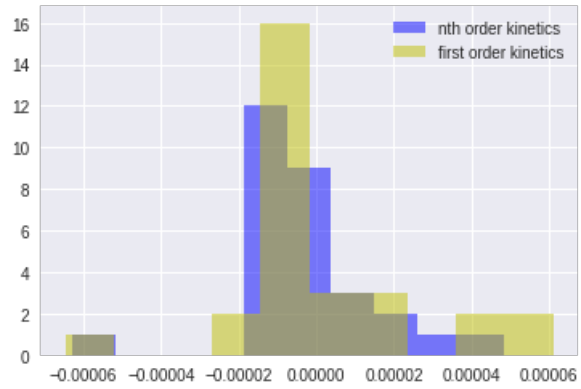


In [22]:

```
plt.hist(r2, color='b', alpha=0.5)
plt.hist(r3, color='y', alpha=0.5)
plt.legend(['nth order kinetics', 'first order kinetics'])
```

Out[22]:

```
<matplotlib.legend.Legend at 0x7fcfdfda3cf8>
```



F-ratio Test

In [23]:

```
from scipy.stats import f

r2 = np.array(r2)
df2 = len(r2) - 1 - 3  # degrees of freedom after fitting 3 parameters

r3 = np.array(r3)
df3 = len(r3) - 1 - 3  # degrees of freedom after fitting 2 parameters

Fratio = r2.var()/r3.var()

pvalue = f.cdf(Fratio, df2, df3)

if pvalue > 0.05:
    print("Cannot reject hypothesis that the variances are equal at p=0.05: Test
pvalue = ", pvalue)
```

```
Cannot reject hypothesis that the variances are equal at p=0.05: Test pvalue =
0.13297014615599262
```

Levene Test

The Levene test provides a more robust test for the null hypothesis that samples are from populations with equal variances.

In [24]:

```
from scipy.stats import levene
levene(r2, r3)
```

Out[24]:

```
LeveneResult(statistic=0.34495202886490245, pvalue=0.5593454146853113)
```

The Levene statistic means that we cannot rule out that the residuals of these models have different variances. Consequently we do not have statistically meaningful evidence that the order of the reaction is different from $n = 1$.

In [ ]:

Chapter 8. Financial Applications

8.1 Binomial Model for Pricing Options

In [ ]:

```
!pip install -q pyomo
!apt-get install -y -qq glpk-utils
```

Historical Data

The first step is download historical data for a selected security or commodity. For the purposes of this notebook, it is useful to choose security of commodities for which there is an active options trading so the pricing model can be compared to real data.

In [5]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd

!wget -N -q "https://raw.githubusercontent.com/jckantor/ND-Pyomo-
Cookbook/master/notebooks/finance/Historical_Adjusted_Close.csv"
S_hist = pd.read_csv('/content/Historical_Adjusted_Close.csv', index_col=0)

S_hist['AAPL'].plot(title='AAPL', logy=True)
plt.ylabel('Adjusted Close')
plt.grid(True)
```

Trim Data Series

In [6]:

```
nYears = 3

S = S_hist['AAPL'].iloc[-nYears*252:]

plt.figure(figsize=(10,4))
S.plot(title=S.name)
plt.ylabel('Adjusted Close')
plt.grid(True)
```

Fitting Historical Data to Geometric Brownian Motion

A model for Geometric Brownian Motion can be written in discrete time as

$$S_{t+\Delta t} = S_t + \mu S_t \Delta t + \sigma S_t \sqrt{\Delta t} Z_t$$

where Z_t is a normal random variate with zero mean and a standard deviation of one. This could also be written as

$$\ln S_{t+\Delta t} = \ln S_t + \nu \Delta t + \sigma \sqrt{\Delta t} Z_t$$

where $\nu = \mu - \frac{\sigma^2}{2}$. These are two different models for price. The difference between μ and ν, i.e. $\frac{\sigma^2}{2}$, is referred to as 'volatility drag.'

Rearranging each of these models

$$r^{Jin} = \frac{S_{t+\Delta t} - S_t}{S_t} = \mu \Delta t + \sigma \sqrt{\Delta t} Z_t \sim \mathcal{N}(\mu \Delta t, \sigma^2 \Delta t)$$
$$r^{log} = \ln S_{t+\Delta t} - \ln S_t = \nu \Delta t + \sigma \sqrt{\Delta t} Z_t \sim \mathcal{N}(\nu \Delta t, \sigma^2 \Delta t)$$

where $\mathcal{N}(\alpha, \beta)$ denotes the normal distribution with mean α and variance β (i.e., standard deviation $\sqrt{\beta}$).

The parameters can be estimated from trading data as

$$\hat{\mu} = \frac{1}{\Delta t} \text{mean}\left(\frac{S_{t+\Delta t} - S_t}{S_t}\right)$$
$$\hat{\nu} = \frac{1}{\Delta t} \text{mean}(\ln S_{t+\Delta t} - \ln S_t)$$
$$\hat{\sigma} = \frac{1}{\sqrt{\Delta t}} \text{stdev}\left(\frac{S_{t+\Delta t} - S_t}{S_t}\right) \approx \frac{1}{\sqrt{\Delta t}} \text{stdev}(\ln S_{t+\Delta t} - \ln S_t)$$

Note that for trading data time is measured in 'trading days'. On average there are 252 trading days in a year.

Period	Trading Days
Year	252
Quarter	63
Month	21
Weekly	4.83
Calendar Day	0.690

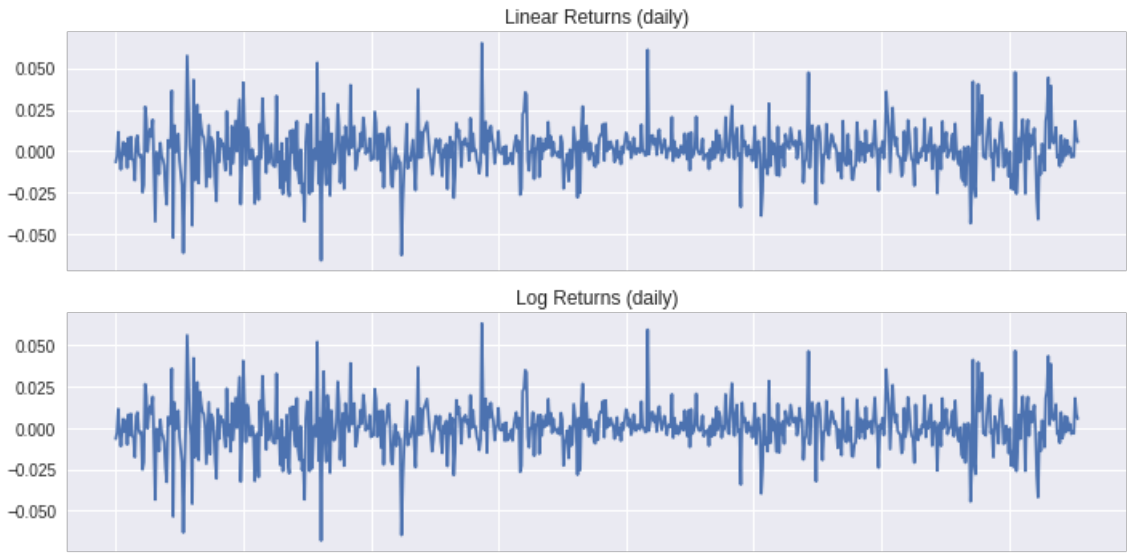
Using these data and the above formulae, μ, ν, and σ can be rescaled to other units of time. For example, if these are estimated from daily trading data, then on an annualized basis

$$\mu^{annual} = 252 \times \mu^{trading\ day}$$
$$\nu^{annual} = 252 \times \nu^{trading\ day}$$
$$\sigma^{annual} = \sqrt{252} \times \sigma^{trading\ day}$$

In [8]:

```
from scipy.stats import norm
import numpy as np

# compute linear and log returns
rlin = (S - S.shift(1))/S.shift(1)
rlog = np.log(S/S.shift(1))

rlin = rlin.dropna()
rlog = rlog.dropna()

# plot data
plt.figure(figsize=(10,5))
plt.subplot(2,1,1)
rlin.plot()
plt.title('Linear Returns (daily)')
plt.tight_layout()

plt.subplot(2,1,2)
rlog.plot()
plt.title('Log Returns (daily)')
plt.tight_layout()

print('\nLinear Returns')
mu,sigma = norm.fit(rlin)
print('   mu = {0:12.8f}  (annualized = {1:.2f}%)'.format(mu,100*252*mu))
print('sigma = {0:12.8f}  (annualized = {1:.2f}%)'.format(sigma,100*np.sqrt(252)*sigma)
)

print('\nLog Returns')
nu,sigma = norm.fit(rlog)
print('   nu = {0:12.8f}  (annualized = {1:.2f}%)'.format(nu,100*252*nu))
print('sigma = {0:12.8f}  (annualized = {1:.2f}%)'.format(sigma,100*np.sqrt(252)*sigma)
)
```

```
Linear Returns
   mu =   0.00071428  (annualized = 18.00%)
sigma =   0.01450301  (annualized = 23.02%)

Log Returns
   nu =   0.00060890  (annualized = 15.34%)
sigma =   0.01450420  (annualized = 23.02%)
```



Binomial Model

The binomial model provides a means of modeling the statistical distribution of future prices. Given a current price S_t, there are two possible states for the next observed value $S_{t+\Delta t}$

$$S_{t+\Delta t} = \begin{cases} uS_t & \text{with probability } p \\ dS_t & \text{with probability } 1-p \end{cases}$$

where u, d, and p are chosen to match the statistics of a model based on Geometric Brownian Motion. The following parameter values are derived in Luenberger (2013),

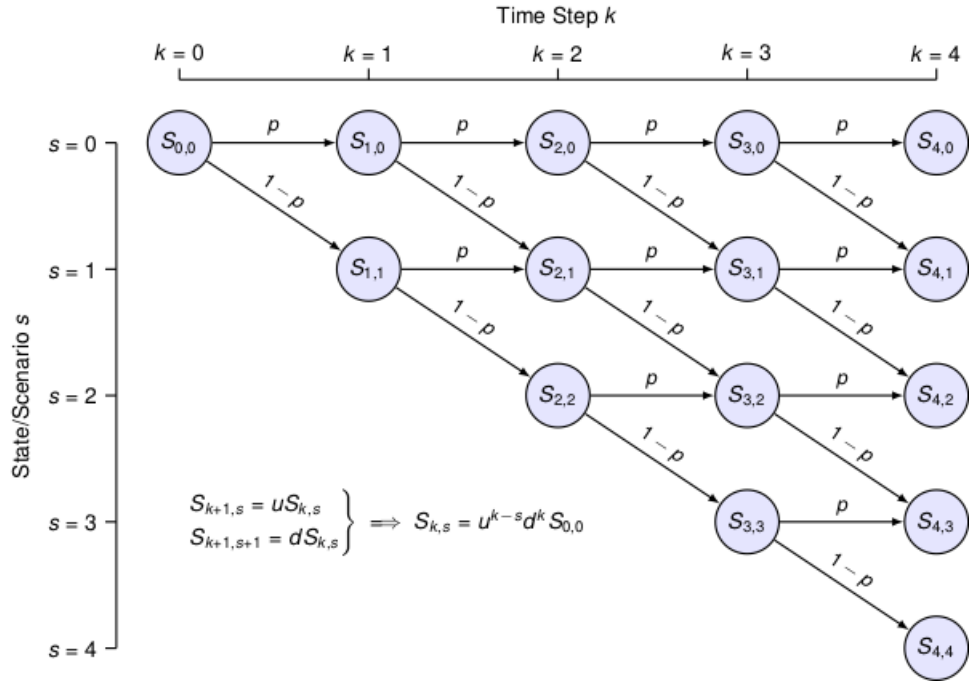
$$p = \frac{1}{2} + \frac{\nu\Delta t}{2\sqrt{\sigma^2\Delta t + (\nu\Delta t)^2}}$$
$$\ln u = \sqrt{\sigma^2\Delta t + (\nu\Delta t)^2}$$
$$\ln d = -\sqrt{\sigma^2\Delta t + (\nu\Delta t)^2}$$

In [9]:

```
# Time/period
dt = 21

p = 0.5 + nu*dt/2/np.sqrt(dt*sigma**2 + (nu*dt)**2)
u = np.exp(np.sqrt(dt*sigma**2 + (nu*dt)**2))
d = np.exp(-np.sqrt(dt*sigma**2 + (nu*dt)**2))

print('Probability (p) = ', round(p,4))
print('  Up Return (u) = ', round(u,4))
print('Down Return (d) = ', round(d,4))
```

```
Probability (p) =  0.5945
  Up Return (u) =  1.07
Down Return (d) =  0.9346
```

The model extends to multiple time steps in a natural way as shown in this diagram:



Note that each step forward in time introduces an additional state to the set of possible outcomes.

For the purpose of coding, we will use Python dictionaries to store future prices S^f. Future prices are indexed by two subscripts, k and s, such that `Sf[k,s]` corresponds to the price at time $t + k\Delta t$ in state s.

We start by setting the initial node equal to the last observed price, $S^f_{0,0} = S_t$. For each k and s there are two subsequent nodes

$$S^f_{k+1,s} = uS^f_{k,s}$$
$$S^f_{k+1,s+1} = dS^f_{k,s}$$

These two equations can be combined by eliminating the common term $S^f_{k,s}$ to yield

$$S^f_{k+1,s} = uS^f_{k,s}$$
$$S^f_{k+1,s+1} = \frac{d}{u} S^f_{k+1,s}$$

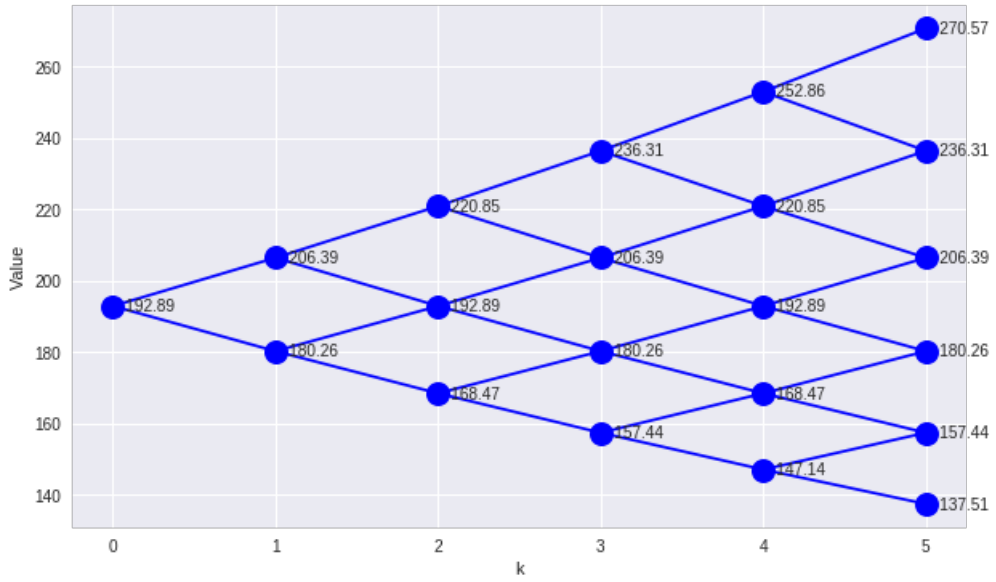
These formula can be solved explicit to give

$$S^f_{k,s} = u^{k-s} d^s S^f_{0,0}$$

which is the formula used below to compute values in the binomial lattice.

In [11]:

```
N = 5

# initialize Sf
Sf = {}
Sf[0,0] = S[-1]

# compute values
for k in range(1,N+1):
    for s in range(0,k+1):
        Sf[k,s] = u**(k-s)*d**s*Sf[0,0]

%matplotlib inline
def Sdisplay(Sf):
    plt.figure(figsize=(10,6))
    for k,s in Sf.keys():
        plt.plot(k,Sf[k,s],'.',ms=30,color='b')
        plt.text(k,Sf[k,s],'    {0:.2f}'.format(Sf[k,s]),ha='left',va='center')
        if (k > 0) & (s < k):
            plt.plot([k-1,k],[Sf[k-1,s],Sf[k,s]],'b')
            plt.plot([k-1,k],[Sf[k-1,s],Sf[k,s+1]],'b')
    plt.xlabel('k')
    plt.ylabel('Value')

Sdisplay(Sf)
```



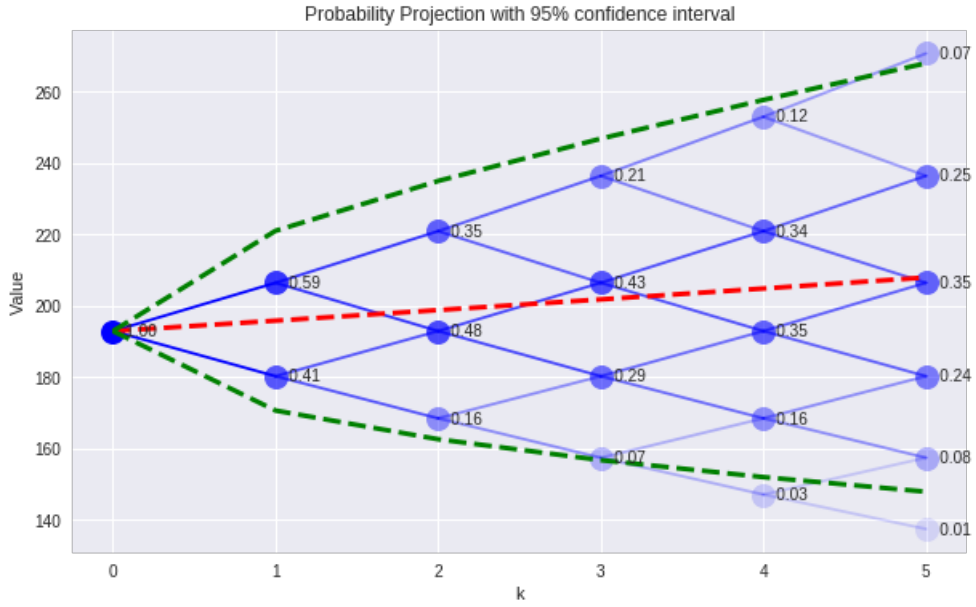
The probability of reaching state s at time step k is denoted by $P_{k,s}$. This can be computed given probability of preceding states and the conditional probabilities p and $1-p$.

$$P_{k,s} = pP_{k-1,s} + (1-p)P_{k-1,s-1}$$

The following cell evaluates price and probability for a binomial model, and also plot the average price.

In [13]:

```
P = {}
P[0,0] = 1

for k in range(0,N):
    P[k+1,0] = p*P[k,0]
    P[k+1,k+1] = (1-p)*P[k,k]
    for s in range(1,k+1):
        P[k+1,s] = p*P[k,s] + (1-p)*P[k,s-1]

%matplotlib inline
def SPdisplay(Sf,P,D):
    plt.figure(figsize=(10,6))
    nPeriods = max([k for k,s in Sf.keys()]) + 1
    Sfmean = np.zeros(N+1)
    Sfvar = np.zeros(N+1)
    for k,s in Sf.keys():
        Sfmean[k] += Sf[k,s]*P[k,s]
        Sfvar[k] += Sf[k,s]**2*P[k,s]
        plt.plot(k,Sf[k,s],'.',ms=30,color='b',alpha=np.sqrt(P[k,s]))
        if (k > 0) & (s < k):
            plt.plot([k-1,k],[Sf[k-1,s],Sf[k,s]],'b',alpha=np.sqrt(P[k-1,s]))
            plt.plot([k-1,k],[Sf[k-1,s],Sf[k,s+1]],'b',alpha=np.sqrt(P[k-1,s]))
    for k,s in D.keys():
        plt.text(k,Sf[k,s],'   {0:.2f}'.format(D[k,s]),ha='left',va='center')
    plt.plot(range(0,N+1),Sfmean,'r--',lw=3)
    Sfstdev = np.sqrt(Sfvar - Sfmean**2)
    plt.plot(range(0,N+1),Sfmean + 1.96*Sfstdev,'g--',lw=3)
    plt.plot(range(0,N+1),Sfmean - 1.96*Sfstdev,'g--',lw=3)
    plt.xlabel('k')
    plt.ylabel('Value')
    plt.title('Probability Projection with 95% confidence interval')
SPdisplay(Sf,P,P)
```



European Call Option

A European Call Option is a contract the provides the holder with the right, but not the obligation, to purchase an asset at a specified price and date in the future. The specified price is generally called the **strike price**, and the specified date is the **expiration date**.

The purpose of the call option is to reduce the holder's exposure to the risk of increasing prices. An airline, for example, might choose to purchase call options on airplance fuels in order to reduce the risk of selling advance tickets.

The value of the call option upon expiration depends on the price of underlying asset. If the asset spot price S is greater than the strike price K, then the call option is worth the difference $S-K$ because that is amount needed to fulfill the contract in the spot market.

On the other hand, if the asset price falls below the strike price, then the option contract has no value since the holder could buy the asset on the spot market for less than the strike price.

$$C_{N,s} = \max(0, S^f_{N,s})$$

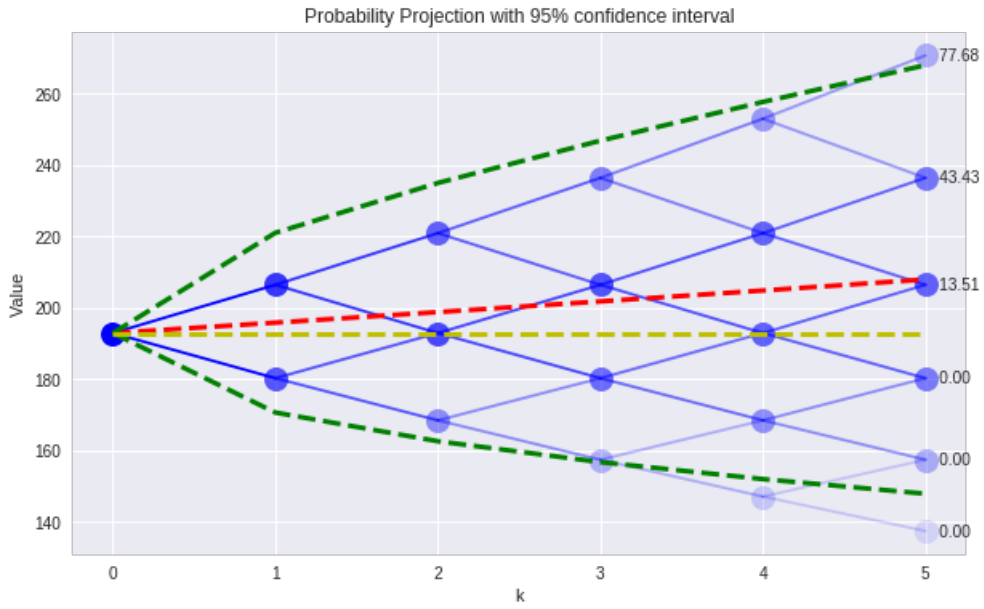
The next cell demonstrates the terminal value of a european call option where the strike price is equal to the initial price (known as an 'at the money' strike).

In [14]:

```
K = S[-1]
C = {}
for s in range(0,N+1):
    C[N,s] = max(0,Sf[N,s] - K)

SPdisplay(Sf,P,C)
plt.plot([0,N],[K,K],'y--',lw=3)
```

Out[14]:

```
[<matplotlib.lines.Line2D at 0x7fec8e64e908>]
```



To price a call option, consider the construction of a portfolio composed of the underlying asset and cash that would have the same payoff. At node k, s, the portfolio is given by

$$C_{k,s} = x_{k,s} S^f_{k,s} + y_{k,s}$$

where $x_{k,s}$ is the number of units of the underlying asset, and $y_{k,s}$ is the cash component of the portfolio. The subsequent value of the portfolio at $k+1$ has two possible values

$$C_{k+1,s} = x_{k,s} u S^f_{k,s} + (1+r) y_{k,s}$$
$$C_{k+1,s+1} = x_{k,s} d S^f_{k,s} + (1+r) y_{k,s}$$

where r is the per period interest rate for cash. Solving for $x_{k,s}$ and $y_{k,s}$,

$$x_{k,s} = \frac{C_{k+1,s} - C_{k+1,s+1}}{(u-d) S^f_{k,s}}$$
$$y_{k,s} = \frac{u C_{k+1,s+1} - d C_{k+1,s}}{(1+r)(u-d)}$$

Inserting these solutions into the original expression,

$$C_{k,s} = \frac{([(1+r)-d] C_{k+1,s} + [u-(1+r)] C_{k+1,s+1}}{(1+r)(u-d)}$$

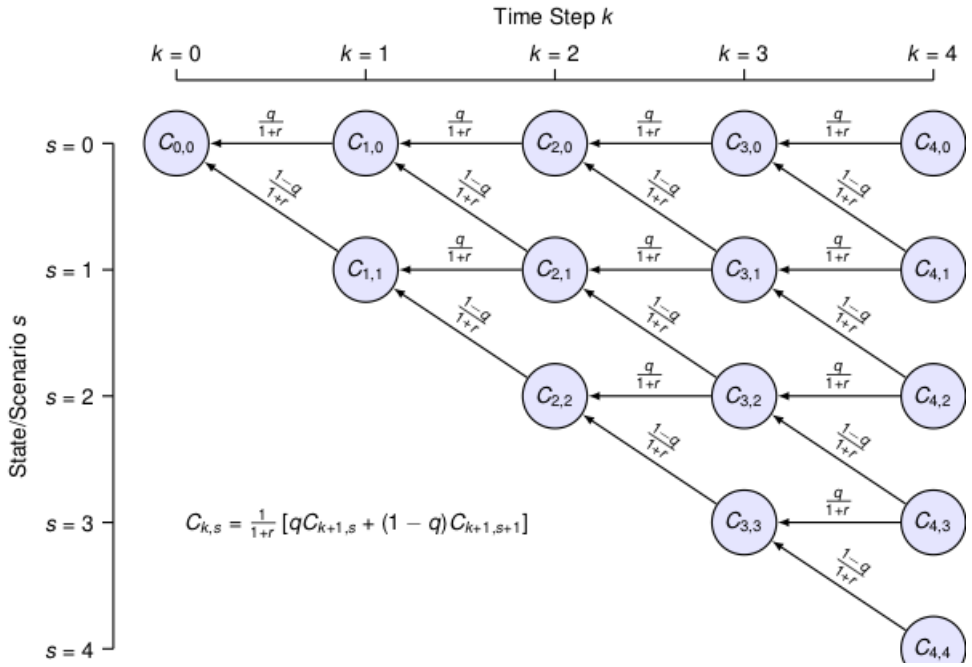
This can be expressed in a far more suggestive form

$$C_{k,s} = \frac{1}{1+r}\left[\underbrace{\frac{(1+r)-d}{u-d}}_{q} C_{k+1,s} + \underbrace{\frac{u-(1+r)}{u-d}}_{1-q} C_{k+1,s+1} \right]$$

or

$$C_{k,s} = \frac{1}{1+r}\left[q C_{k+1,s} + (1-q) C_{k+1,s+1} \right]$$

This expression has two important consequences. The first is an interpretation as a 'risk-neutral' probability such that $C_{k,s}$ is the 'expected value' of the call option. This is not the real-world probability! Instead, it is different measure that provides a very useful means of computing the value of options as illustrated in the following diagram.



The second important consequence is the actual pricing formula. It's linear, for one thing, is easily implemented as a calculation that proceeds backward in time as shown in the diagram above. As can be seen from the derivation, the computation is quite general and can be applied to any option which can be assigned a terminal value.

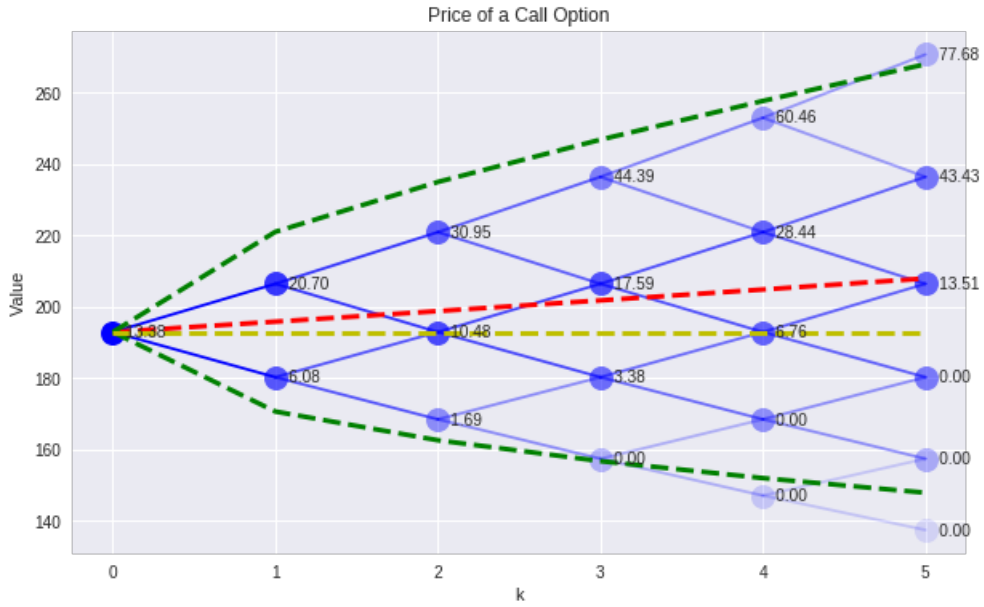
In [15]:

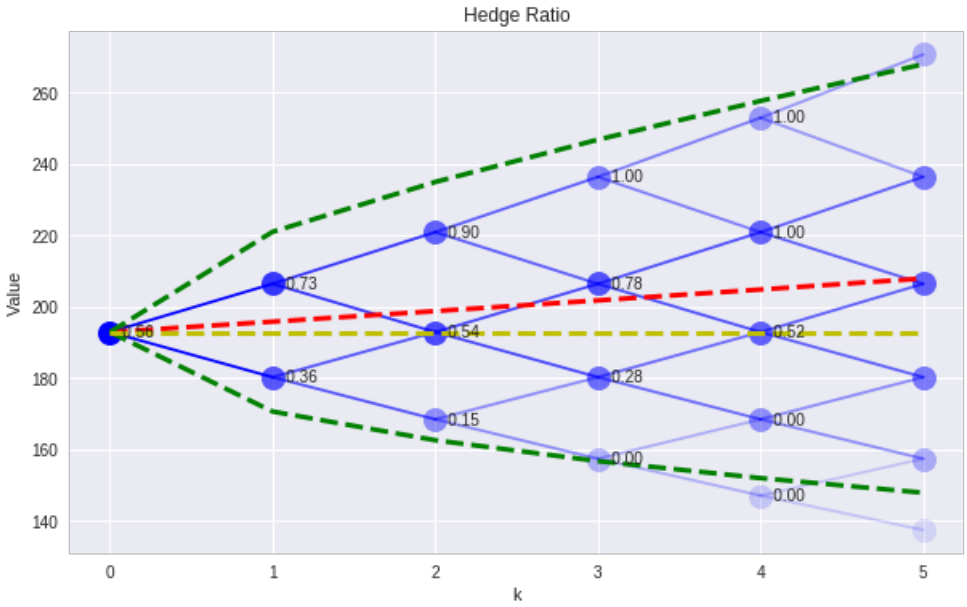
```
r = 0.030/12
q = (1+r-d)/(u-d)
print('q = ',q)

C = {}
x = {}
y = {}

for s in range(0,N+1):
    C[N,s] = max(0,Sf[N,s] - K)

for k in reversed(range(0,N)):
    for s in range(0,k+1):
        C[k,s] = (q*C[k+1,s]+(1-q)*C[k+1,s+1])/(1+r)
        x[k,s] = (C[k+1,s]-C[k+1,s+1])/(u-d)/Sf[k,s]
        y[k,s] = C[k,s] - x[k,s]*Sf[k,s]

SPdisplay(Sf,P,C)
plt.plot([0,N],[K,K],'y--',lw=3)
plt.title('Price of a Call Option')

SPdisplay(Sf,P,x)
plt.plot([0,N],[K,K],'y--',lw=3)
plt.title('Hedge Ratio')

SPdisplay(Sf,P,y)
plt.plot([0,N],[K,K],'y--',lw=3)
plt.title('Cash Position')
```

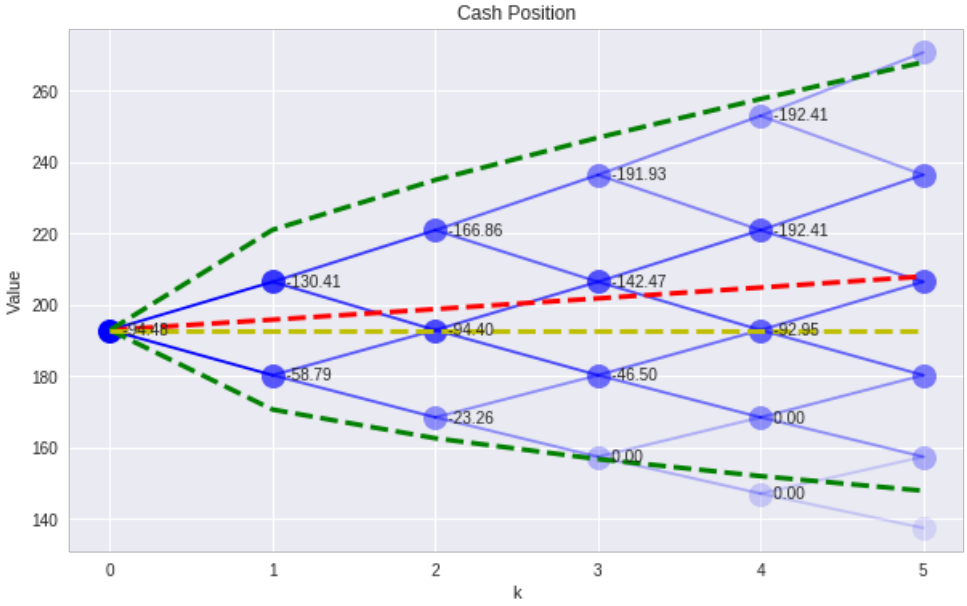
```
q =  0.5015387932713019
```

Out[15]:

```
Text(0.5,1,'Cash Position')
```




Hedge Ratio
Value
k
260
240
220
200
180
160
140
0
1
2
3
4
5
1.00
1.00
0.90
1.00
0.73
0.78
0.56
0.54
0.52
0.36
0.28
0.15
0.00
0.00
0.00


Cash Position
Value
k
260
240
220
200
180
160
140
0
1
2
3
4
5
-192.41
-191.93
-166.86
-192.41
-130.41
-142.47
-94.40
-92.95
-58.79
-46.50
-23.26
0.00
0.00
0.00

Implementing a Replicating Portfolio with Pyomo

A replicating portfolio is the key concept that allows use of the binomial model for pricing options. In a nutshell, the value of an option is the money needed to construct a portfolio exactly replicate the option payoff.

This concept can be extended to applications involving 'real assets', including processes that convert commodity resources into higher value products. As a first step, let's see how Pyomo can be used to model a replicating portfolio for a European call option.

We start with an expression for the value of the replicating portfolio at time k in state s. For the purpose of later generalization, we use the symbol W to denote wealth,

$$W_{k,s} = x_{k,s} S^f_{k,s} + y_{k,s} B_k$$

The portfolio consists of $x_{k,s}$ units of an underlying asset with a price $S^f_{k,s}$ subject to statistical variability, and $y_{k,s}$ units of a 'bond' which has a known future price and therefore depends only on k. For example,

$$B_k = (1 + r_f)^k B_0$$

where r_f is a risk-free interest rate. This is the 'cash' component of the portfolio.

For a European call option we must have enough wealth on-hand to pay off the value of the call option at $k = N$. This requirement can be written as inequalities

$$W_{N,s} \geq \max(0, S^f_{N,s} - K) \qquad \forall s \in \mathcal{S}_N$$

where $\mathcal{S}_N$ is the set of possible states at time step N.

At earlier nodes, the value of the portfolio must be sufficient to finance the portfolio as subsequent states. In the binomial model, for each node (k, s) there are two subsequenct states $(k+1, s)$ and $(k+1, s+1)$. This results in two constraints:

$$x_{k,s} S^f_{k+1,s} + y_{k,s} B_{k+1} \geq W_{k+1,s}$$
$$x_{k,s} S^f_{k+1,s+1} + y_{k,s} B_{k+1} \geq W_{k+1,s+1}$$

What we seek is the minimum cost portfolio at the initial node, i.e.,

$$\min_{x,y} W_{0,0}$$

subject to the above constraints.

If everything works as expected, the results of this calculation should be the same as those computed using risk-neutral probabilities.

In [ ]:

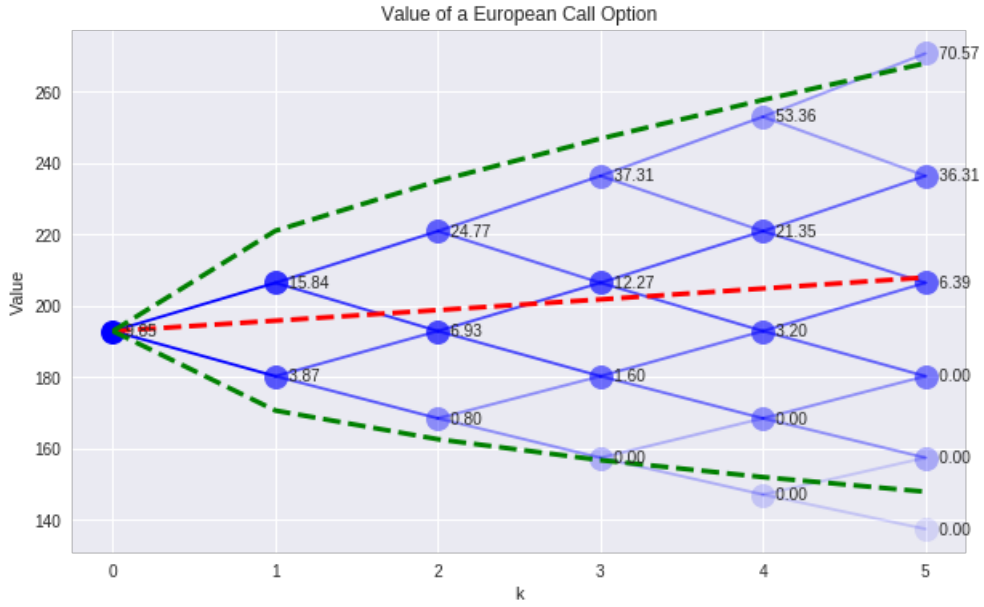
```
K = 200
```

In [17]:

```
from pyomo.environ import *

# set of periods and states for each period
Periods = range(0,N+1)
States = range(0,N+1)

# future bond prices
B = [(1+r)**k for k in Periods]

m = ConcreteModel()

# model variables
m.W = Var(Periods,States,domain=Reals)
m.x = Var(Periods,States,domain=Reals)
m.y = Var(Periods,States,domain=Reals)

# objective
m.OBJ = Objective(expr = m.W[0,0], sense=minimize)

# constraint list
m.cons = ConstraintList()

for k in Periods:
    for s in range(0,k+1):
        m.cons.add(m.W[k,s] == m.x[k,s]*Sf[k,s] + m.y[k,s]*(1+r)**k)

for s in States:
    m.cons.add(m.W[N,s] >= max(0,Sf[N,s] - K))

for k in range(0,N):
    for s in range(0,k+1):
        m.cons.add(m.x[k,s]*Sf[k+1,s] + m.y[k,s]*B[k+1] >= m.W[k+1,s])
        m.cons.add(m.x[k,s]*Sf[k+1,s+1] + m.y[k,s]*B[k+1] >= m.W[k+1,s+1])

SolverFactory('glpk').solve(m)

W = {}
for k in range(0,N+1):
    for s in range(0,k+1):
        W[k,s] = m.W[k,s]()

SPdisplay(Sf,P,W)
plt.title('Value of a European Call Option')
```

Out[17]:

```
Text(0.5,1,'Value of a European Call Option')
```



European Put Option

A European put option is the right (without obligation) to sell an asset for specified price on a specified date. The analysis is proceeds exactly as for the European call option, except that the value on the terminal date is given by

$$P_{N,s} = \max(0, K - S^f_{N,s}) \qquad \forall s \in \mathcal{S}_N$$

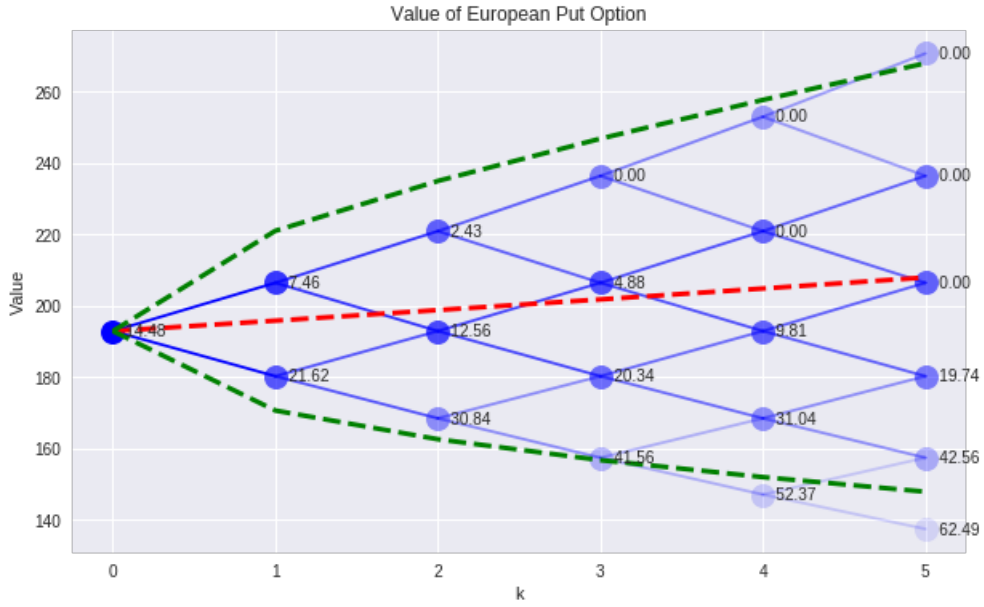
The value of the put option can be computed using the same risk neutral probabilities as for the call option, or by modifying the Pyomo model as shown in the following cell.

In [18]:

```
from pyomo.environ import *

# set of periods and states for each period
Periods = range(0,N+1)
States = range(0,N+1)

# future bond prices
B = [(1+r)**k for k in Periods]

m = ConcreteModel()

# model variables
m.W = Var(Periods,States,domain=Reals)
m.x = Var(Periods,States,domain=Reals)
m.y = Var(Periods,States,domain=Reals)

# objective
m.OBJ = Objective(expr = m.W[0,0], sense=minimize)

# constraint list
m.cons = ConstraintList()

for k in Periods:
    for s in range(0,k+1):
        m.cons.add(m.W[k,s] == m.x[k,s]*Sf[k,s] + m.y[k,s]*(1+r)**k)

for s in States:
    m.cons.add(m.W[N,s] >= max(0,K - Sf[N,s]))

for k in range(0,N):
    for s in range(0,k+1):
        m.cons.add(m.x[k,s]*Sf[k+1,s] + m.y[k,s]*B[k+1] >= m.W[k+1,s])
        m.cons.add(m.x[k,s]*Sf[k+1,s+1] + m.y[k,s]*B[k+1] >= m.W[k+1,s+1])

SolverFactory('glpk').solve(m)

W = {}
for k in range(0,N+1):
    for s in range(0,k+1):
        W[k,s] = m.W[k,s]()

SPdisplay(Sf,P,W)
plt.title('Value of European Put Option')
```

Out[18]:

```
Text(0.5,1,'Value of European Put Option')
```



Early Exercise

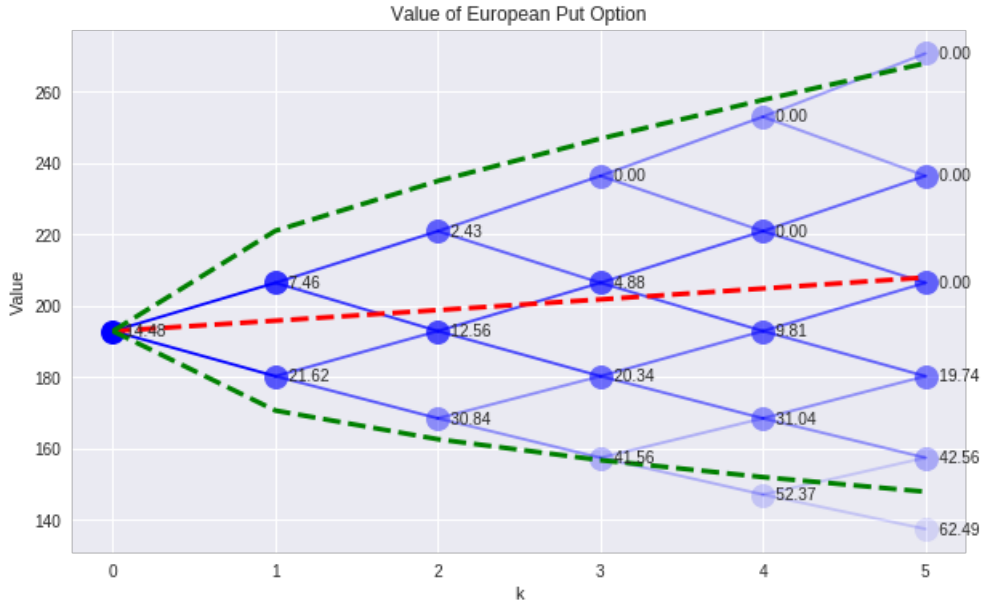
Let's compare the value of the put option to the value of exercising the option. At any node (k, s), the value of exercising the option is $K - S^f_{k,s}$. The next cell compares the value of the European put option to hypothetical value of early exercise.

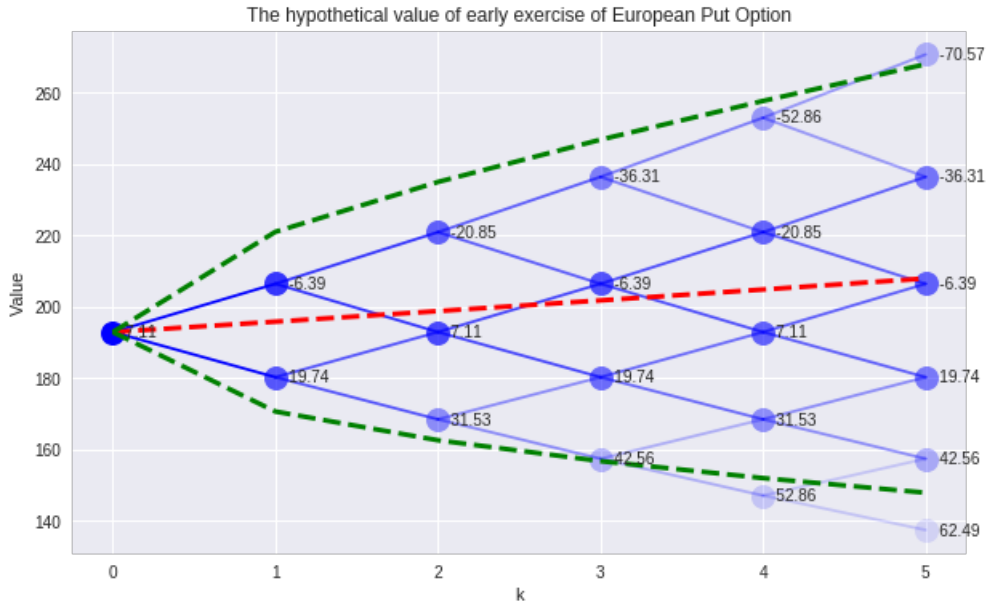
In [19]:

```
W = {}
for k in range(0,N+1):
    for s in range(0,k+1):
        W[k,s] = m.W[k,s]()

SPdisplay(Sf,P,W)
plt.title('Value of European Put Option')

E = {}
for k in range(0,N+1):
    for s in range(0,k+1):
        E[k,s] = K - Sf[k,s]

SPdisplay(Sf,P,E)
plt.title('The hypothetical value of early exercise of European Put Option')
```

Out[19]:

```
Text(0.5,1,'The hypothetical value of early exercise of European Put Option')
```





We see several nodes where the value of exercising the option is worth more than the option itself. These would be opportunities for a risk-free profit. If such a circumstance arises, then

1. buy the put option at price $P_{k,s}$
2. buy the asset at price $S^f_{k,s}$
3. exercise the option to sell at the strike price K

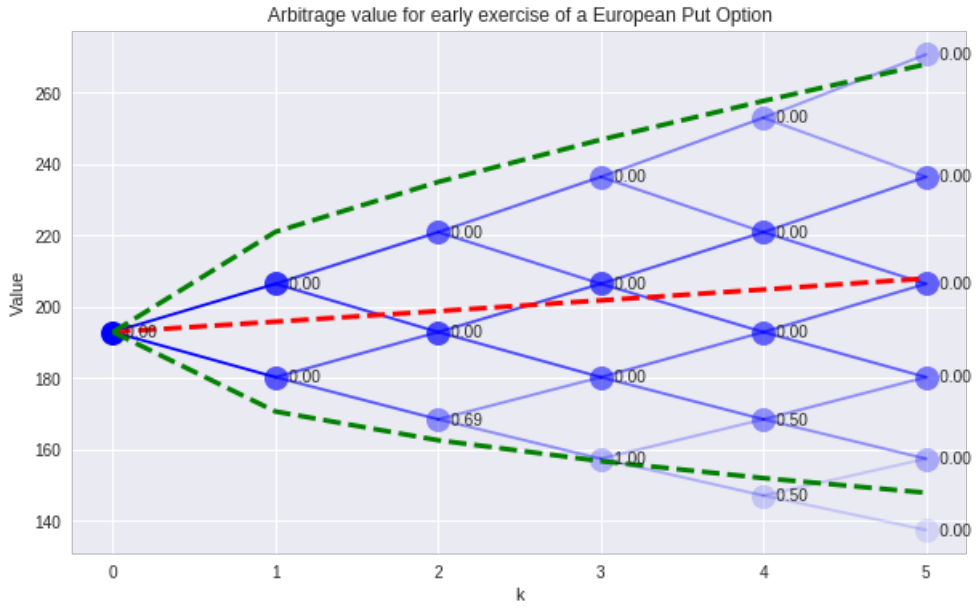
If $K > P_{k,s} + S^f_{k,s}$ then this deal offers a completely risk-free profit.

In [20]:

```
Arbitrage = {}
for k in range(0,N+1):
    for s in range(0,k+1):
        Arbitrage[k,s] = max(0,(K - Sf[k,s] - m.W[k,s]()))

SPdisplay(Sf,P,Arbitrage)
plt.title('Arbitrage value for early exercise of a European Put Option')
```

Out[20]:

```
Text(0.5,1,'Arbitrage value for early exercise of a European Put Option')
```



American Put Option

Compared to a European option, an American option provides an additional right of early exercise. The holder of the option an choose to exercise the option at any point prior to the expiration date. This additional right can (but does not always add value under certain conditions.

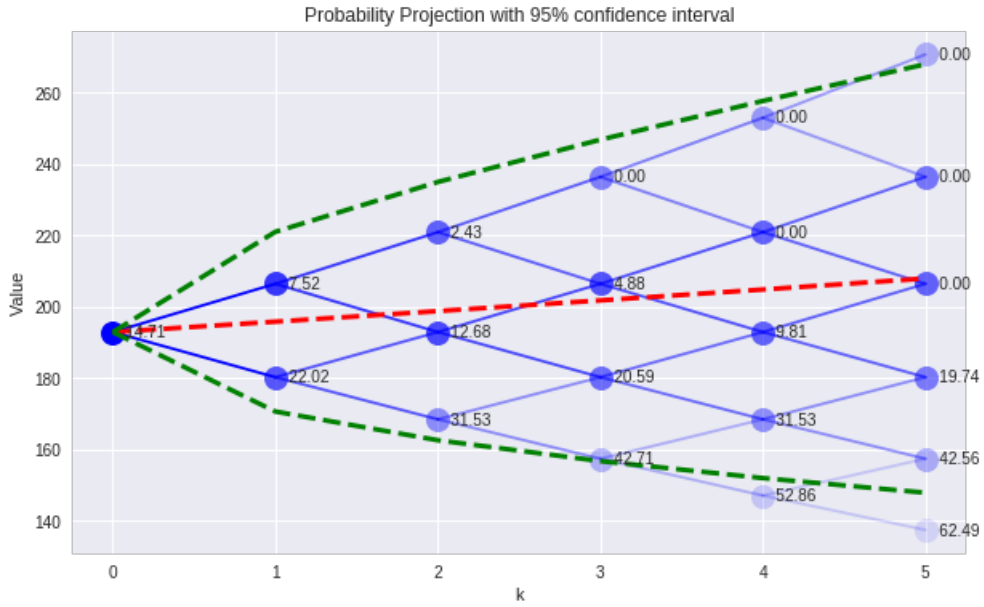
The opportunity for early exerise adds some complexity to the calculation of value. In the optimization framework, the holder of an American put could always receive value by exercising the option if the market value of the option is less than the exercise value. Thus the exercise value establishes a lower bound on the value of the option at all nodes (not just the terminal nodes)

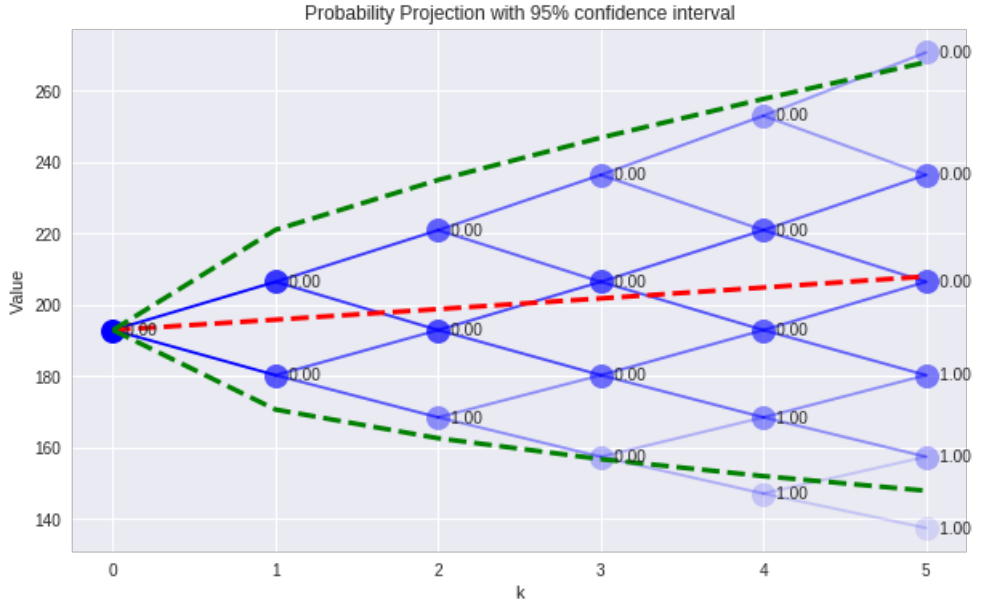
$$P_{k,s} \geq K - S^f_{k,s}$$

This is demonstrated in the following cell.

In [21]:

```
from pyomo.environ import *

# set of periods and states for each period
Periods = range(0,N+1)
States = range(0,N+1)

# future bond prices
B = [(1+r)**k for k in Periods]

m = ConcreteModel()

# model variables
m.W = Var(Periods,States,domain=Reals)
m.x = Var(Periods,States,domain=Reals)
m.y = Var(Periods,States,domain=Reals)

# objective
m.OBJ = Objective(expr = m.W[0,0], sense=minimize)

# constraint list
m.cons = ConstraintList()

for k in Periods:
    for s in range(0,k+1):
        m.cons.add(m.W[k,s] == m.x[k,s]*Sf[k,s] + m.y[k,s]*(1+r)**k)

for k in Periods:
    for s in range(0,k+1):
        m.cons.add(m.W[k,s] >= max(0,K - Sf[k,s]))

for k in range(0,N):
    for s in range(0,k+1):
        m.cons.add(m.x[k,s]*Sf[k+1,s] + m.y[k,s]*B[k+1] >= m.W[k+1,s])
        m.cons.add(m.x[k,s]*Sf[k+1,s+1] + m.y[k,s]*B[k+1] >= m.W[k+1,s+1])

SolverFactory('glpk').solve(m)

W = {}
E = {}
for k in range(0,N+1):
    for s in range(0,k+1):
        W[k,s] = m.W[k,s]()
        E[k,s] = np.abs(K-Sf[k,s] - m.W[k,s]()) < 0.01

SPdisplay(Sf,P,W)

SPdisplay(Sf,P,E)
```


Probability Projection with 95% confidence interval
Value
k
0.00
0.00
0.00
0.00
0.00
0.00
2.43
7.52
4.88
4.71
12.68
9.81
22.02
20.59
19.74
31.53
31.53
42.71
42.56
52.86
62.49


Probability Projection with 95% confidence interval
Value
k
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
1.00
1.00
1.00
1.00
1.00

In [ ]:

8.2 Historical Stock Data

In [1]:

```
data_dir = 'data/stocks'

djia = ['AXP','BA','CAT','CSCO','CVX','DD','DIS','GE', \
        'GS','HD','IBM','INTC','JNJ','JPM','KO','MCD', \
        'MMM','MRK','MSFT','NKE','PFE','PG','T','TRV', \
        'UNH','UTX','V','VZ','WMT','XOM']

favs = ['AAPL']

stocks = djia + favs

import os
os.makedirs(data_dir, exist_ok=True)
```

Alpha Vantage

The following cells retrieve a history of daily trading data for a specified set of stock ticker symbols. These functions use the free Alpha Vantage data service.

The service requires an personal api key which can be claimed here in just a few seconds. Place the key as a string in a file `data/api_key.txt` in the data directory as this notebook (note: api_key.txt is not distributed with the github repository). The function `api_key()` returns the key stored in `api_key.txt`.

In [2]:

```
def api_key():
    "Read api_key.txt and return api_key"
    try:
        with open('data/api_key.txt') as fp:
            line = fp.readline()
    except:
        raise RuntimeError('Error while attempting to read data/api_key.txt')
    return line.strip()
```

The function `alphavantage(s)` returns a pandas dataframe holding historical trading data for a stocker ticker symbol specified by `s`.

In [3]:

```
import os
import requests
import pandas as pd

def alphavantage(symbol=None):
    if symbol is None:
        raise ValueError("No symbol has been provided")
    payload = {
        "function": "TIME_SERIES_DAILY_ADJUSTED",
        "symbol": symbol,
        "outputsize": "full",
        "datatype": "json",
        "apikey": api_key(),
    }
    api_url = "https://www.alphavantage.co/query"
    try:
        response = requests.get(api_url, params=payload)
    except:
        raise ValueError("No response using api key: " + api_key)
    data = response.json()
    k = list(data.keys())
    metadata = data[k[0]]
    timeseries = data[k[1]]
    S = pd.DataFrame.from_dict(timeseries).T
    S = S.apply(pd.to_numeric)
    S.columns = [h.lstrip('12345678. ') for h in S.columns]
    return S

alphavantage('AAPL').head()
```

Out[3]:

	open	high	low	close	adjusted close	volume	dividend amount	split coefficient
2000-01-03	104.87	112.50	101.69	111.94	3.5554	4783900	0.0	1.0
2000-01-04	108.25	110.62	101.19	102.50	3.2556	4574800	0.0	1.0
2000-01-05	103.75	110.56	103.00	104.00	3.3032	6949300	0.0	1.0
2000-01-06	106.12	107.00	95.00	95.00	3.0174	6856900	0.0	1.0
2000-01-07	96.50	101.00	95.50	99.50	3.1603	4113700	0.0	1.0

`get_stock_data(symbols)` retrieves trading data for a list of symbols and stores each in seperate file in the data directory. The file name is the ticker symbol with a `.csv` suffix.

In [4]:

```
def get_stock_data(symbols, service=alphavantage):
    if isinstance(symbols, str):
        symbols = [symbols]
    assert all(isinstance(s, str) for s in symbols)
    for s in symbols:
        print('downloading', s, end='')
        k = 3
        while k > 0:
            try:
                k -= 1
                S = service(s)
                S.to_csv(os.path.join(data_dir, s + '.csv'))
                print(' success')
                break
            except:
                print(' fail', end='')
            print('')

get_stock_data(['AAPL'])
```

```
downloading AAPL success
```

Download Selected Ticker Symbols

In [5]:

```
get_stock_data(stocks)
```

```
downloading AXP success
downloading BA success
downloading CAT success
downloading CSCO success
downloading CVX success
downloading DD fail
 fail
 fail
downloading DIS success
downloading GE success
downloading GS success
downloading HD success
downloading IBM success
downloading INTC success
downloading JNJ success
downloading JPM success
downloading KO success
downloading MCD success
downloading MMM success
downloading MRK success
downloading MSFT success
downloading NKE success
downloading PFE success
downloading PG success
downloading T success
downloading TRV success
downloading UNH success
downloading UTX success
downloading V success
downloading VZ success
downloading WMT success
downloading XOM success
downloading AAPL success
```

In [ ]:

```
```

8.3 Charting Stock Data

Loading Stock Data from Data Directory

In [5]:

```
import os
import pandas as pd

data_dir = 'data/stocks'

stocks = {}
for file in sorted(os.listdir(data_dir)):
    if file.endswith(".csv"):
        s = file.split('.')[0]
        fname = os.path.join(data_dir, file)
        stocks[s] = pd.read_csv(fname, index_col=0)
```

In [6]:

```
stocks.keys()
```

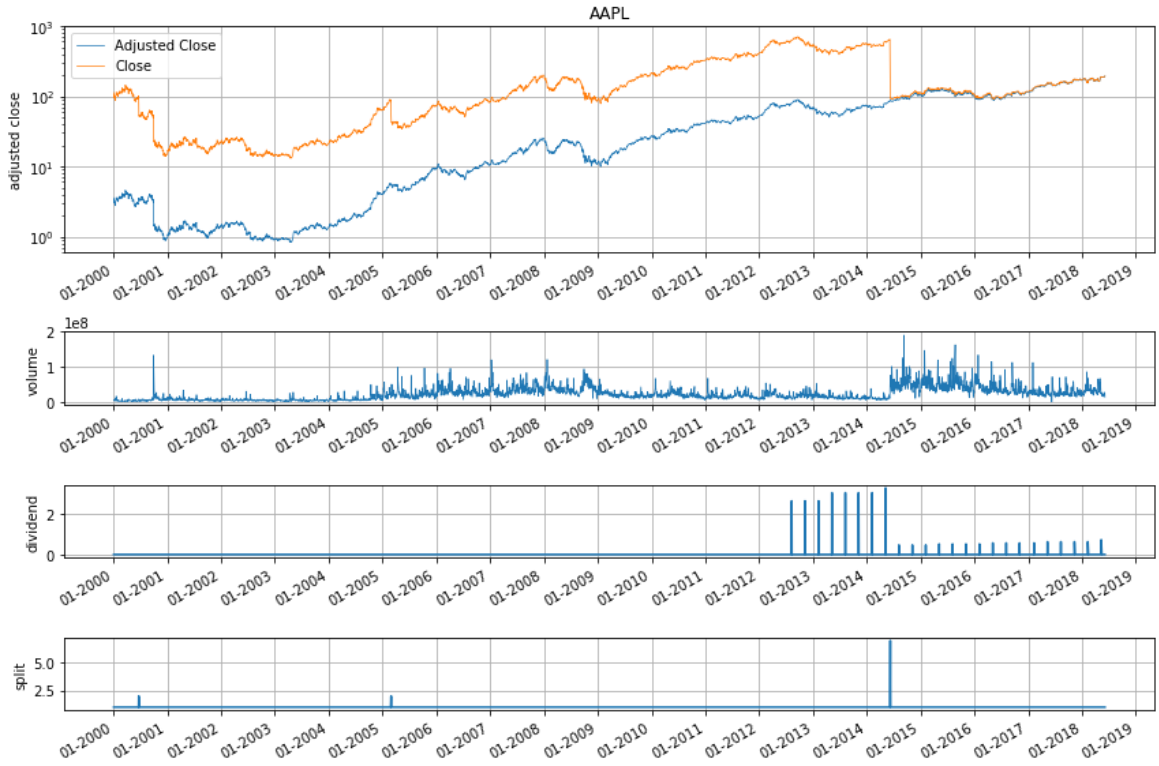
Out[6]:

```
dict_keys(['AAPL', 'AXP', 'BA', 'CAT', 'CSCO', 'CVX', 'DD', 'DIS', 'F', 'GE', 'GS',
'HD', 'IBM', 'INTC', 'JNJ', 'JPM', 'KO', 'MCD', 'MMM', 'MRK', 'MSFT', 'NKE', 'PFE', 'P
G', 'T', 'TRV', 'UNH', 'UTX', 'V', 'VZ', 'WMT', 'XOM'])
```

Charting

In [7]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

def stock_chart(symbol):
    S = pd.DataFrame.from_dict(stocks[symbol])
    S.index = pd.to_datetime(S.index)
    plt.figure(figsize=(12,8))

    ax = plt.subplot(5,1,(1,2))
    S['adjusted close'].plot(ax=ax, lw=0.7, logy=True)
    S['close'].plot(ax=ax, lw=0.7, logy=True)
    plt.legend(['Adjusted Close','Close'])
    plt.title(symbol)
    plt.ylabel('adjusted close')
    ax.xaxis.set_major_locator(mdates.YearLocator())
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
    plt.grid()

    ax = plt.subplot(5,1,3)
    S['volume'].plot(lw=0.7)
    plt.ylabel('volume')
    plt.tight_layout()
    ax.xaxis.set_major_locator(mdates.YearLocator())
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
    plt.grid()

    ax = plt.subplot(5,1,4)
    S['dividend amount'].plot()
    ax.xaxis.set_major_locator(mdates.YearLocator())
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
    plt.ylabel('dividend')
    plt.grid()

    ax = plt.subplot(5,1,5)
    S['split coefficient'].plot()
    ax.xaxis.set_major_locator(mdates.YearLocator())
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
    plt.ylabel('split')
    plt.grid()

    plt.tight_layout()

stock_chart('AAPL')
```

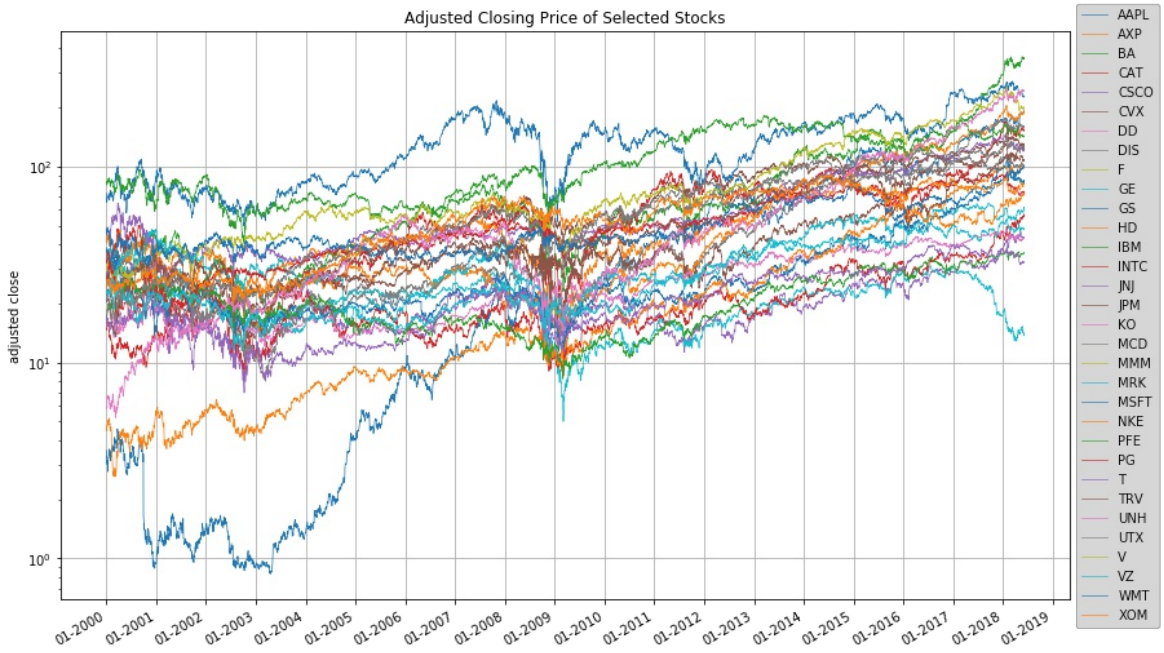


Consolidating Adjusted Close Data

In [8]:

```
S = pd.concat([stocks[s]['adjusted close'] for s in stocks.keys()], axis=1, keys=stocks
.keys())
S.index = pd.to_datetime(S.index)

fig, ax = plt.subplots(figsize=(14,9))
S.plot(ax=ax, lw=0.7, logy=True)
ax.xaxis.set_major_locator(mdates.YearLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
plt.ylabel('adjusted close')
plt.title('Adjusted Closing Price of Selected Stocks')
plt.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))
plt.grid()

S.to_csv('Historical_Adjusted_Close.csv')
```

```
/Users/jeff/anaconda3/lib/python3.6/site-packages/matplotlib/scale.py:111:
RuntimeWarning: invalid value encountered in less_equal
  out[a <= 0] = -1000
```



In [ ]:

In [ ]:

In [ ]:

8.4 MAD Portfolio Optimization

In [ ]:

```
%%capture
!pip install -q pyomo
!apt-get install -y -qq glpk-utils

!pip install ipywidgets
```

In [ ]:

```
%matplotlib inline
import numpy as np
import random
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import pandas as pd
```

Investment Objectives

- Maximize returns
- Reduce Risk through diversification

Why Diversify?

Investment portfolios are collections of investments that are managed for overall investment return. Compared to investing all of your capital into a single asset, maintaining a portfolio of investments allows you to manage risk through diversification.

Reduce Risk through Law of Large Numbers

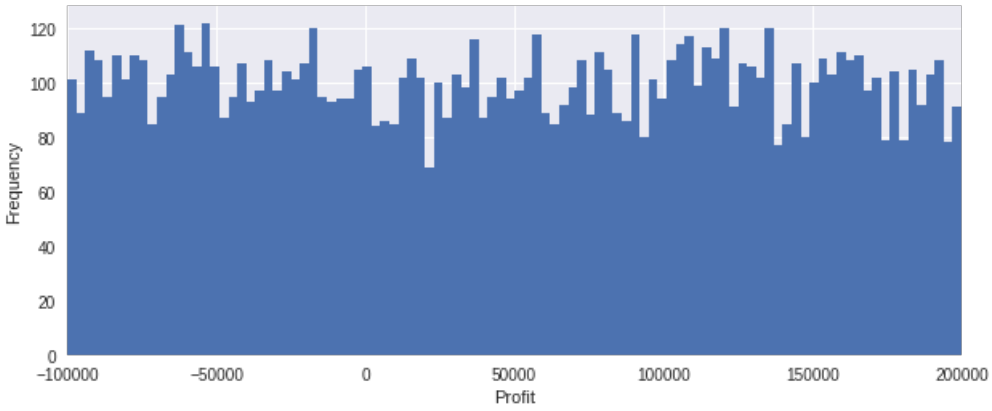
Suppose there are a set of independent investment opportunities that will pay back between 0 and 300% of your original investment, and that all outcomes in that range are equally likely. You have $100,000 to invest. Should you put it all in one opportunity? Or should you spread it around?

Here we simulate the outcomes of 1000 trials where we place all the money into a sigle investment of $100,000.

In [5]:

```
W0 = 100000.00

Ntrials = 10000
Profit = list()
for n in range(0, Ntrials):
    W1 = W0*random.uniform(0,3.00)
    Profit.append(W1 - W0)

plt.figure(figsize=(10,4))
plt.hist(Profit, bins=100)
plt.xlim(-100000, 200000)
plt.xlabel('Profit')
plt.ylabel('Frequency')

print('Average Profit = ${:.0f}'.format(np.mean(Profit)))
```

```
Average Profit = $49101
```



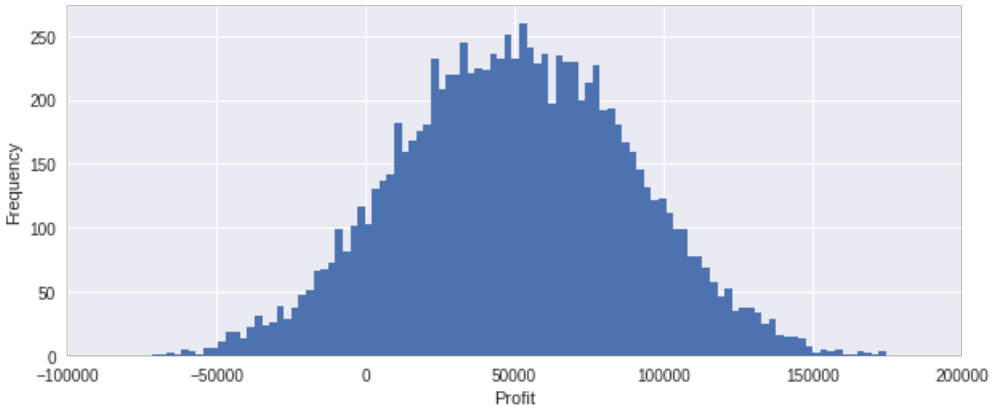
As you would expect, about 1/3 of the time there is a loss, and about 2/3 of the time there is a profit. In the extreme we can lose all of our invested capital. Is this an acceptable investment outcome?

Now let's see if what happens if we diversify our investment. We'll assume the investment outcomes have exactly the same probabilities. The only difference is that instead of one big investment of $100,000, we'll break our capital into 5 smaller sized investments of $20,000 each. We'll calculate the probability distribution of outcomes.

In [6]:

```
W0 = 100000.00

Ntrials = 10000
Ninvestments = 5

Profit = list()
for n in range(0,Ntrials):
    W1 = sum([(W0/Ninvestments)*random.uniform(0,3.00) for _ in range(0,Ninvestments)])
    Profit.append(W1-W0)

plt.figure(figsize=(10,4))
plt.hist(Profit, bins=100)
plt.xlim(-100000, 200000)
plt.xlabel('Profit')
plt.ylabel('Frequency')

print('Average Profit = ${:.0f}'.format(np.mean(Profit)))
```

```
Average Profit = $50272
```



What we observe is that even a small amount of diversification can dramatically reduce the downside risk of experiencing a loss. We also see the upside potential has been reduced. What hasn't changed is the that average profit remains at $50,000. Whether or not the loss of upside potential in order to reduce downside risk is an acceptable tradeoff depends on your individual attitude towards risk.

Value at Risk (VaR)

Value at risk (VaR) is a measure of investment risk. Given a histogram of possible outcomes for the profit of a portfolio, VaR corresponds to negative value of the 5th percentile. That is, 5% of all outcomes would have a lower outcome, and 95% would have a larger outcome.

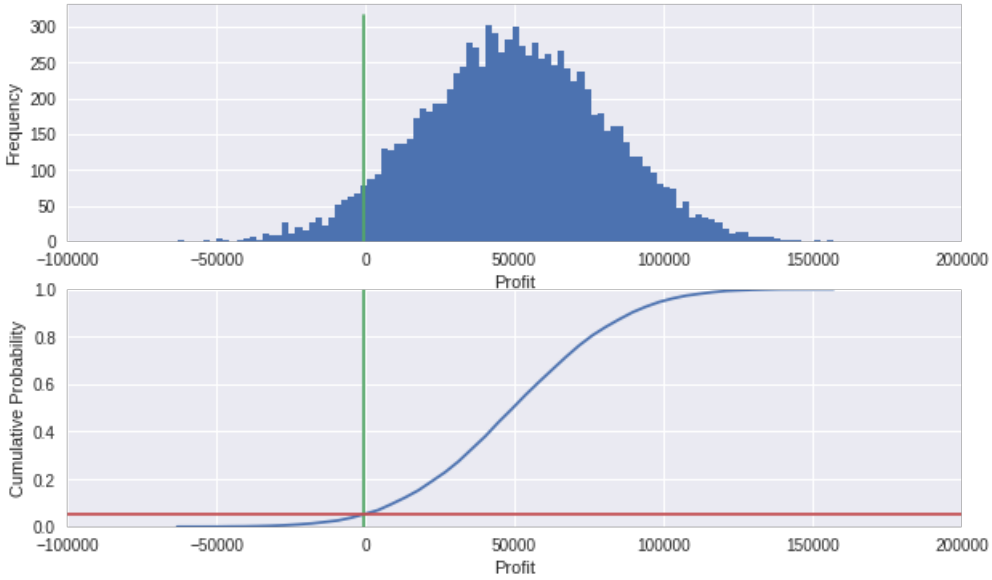
The conditional value at risk (also called the expected shortfall (ES), average value at risk (aVaR), and the expected tail loss (ETL)) is the negative of the average value of the lowest 5% of outcomes.

The following cell provides an interactive demonstration. Use the slider to determine how to break up the total available capital into a number of smaller investments in order to reduce the value at risk to an acceptable (to you) level. If you can accept only a 5% probability of a loss in your portfolio, how many individual investments would be needed?

In [19]:

```
#@title Value at Risk (VaR) Demo { run: "auto", vertical-output: true }
Ninvestments = 8 #@param {type:"slider", min:1, max:20, step:1}

from statsmodels.distributions import ECDF

W0 = 100000.00
Ntrials = 10000

def sim(Ninvestments = 5):

    Profit = list()
    for n in range(0, Ntrials):
        W1 = sum([(W0/Ninvestments)*random.uniform(0,3.00) for _ in
range(0,Ninvestments)])
        Profit.append(W1-W0)

    print('Average Profit = ${:.0f}'.format(np.mean(Profit)).replace('$-','-$'))

    VaR = -sorted(Profit)[int(0.05*Ntrials)]
    print('Value at Risk (95%) = ${:.0f}'.format(VaR).replace('$-','-$'))

    cVaR = -sum(sorted(Profit)[0:int(0.05*Ntrials)])/(0.05*Ntrials)
    print('Conditional Value at Risk (95%) = ${:.0f}'.format(cVaR).replace('$-','-$'))

    plt.figure(figsize=(10,6))
    plt.subplot(2,1,1)
    plt.hist(Profit, bins=100)
    plt.xlim(-100000, 200000)
    plt.plot([-VaR, -VaR], plt.ylim())
    plt.xlabel('Profit')
    plt.ylabel('Frequency')

    plt.subplot(2,1,2)
    ecdf = ECDF(Profit)
    x = np.linspace(min(Profit), max(Profit))
    plt.plot(x,ecdf(x))
    plt.xlim(-100000, 200000)
    plt.ylim(0,1)
    plt.plot([-VaR, -VaR], plt.ylim())
    plt.plot(plt.xlim(), [0.05, 0.05])
    plt.xlabel('Profit')
    plt.ylabel('Cumulative Probability');

sim(Ninvestments)
```

```
Average Profit = $49560
Value at Risk (95%) = $797
Conditional Value at Risk (95%) = $13373
```



Import Historical Stock Price Data

In [24]:

```
!wget -N -q "https://raw.githubusercontent.com/jckantor/ND-Pyomo-
Cookbook/master/notebooks/finance/Historical_Adjusted_Close.csv"
S_hist = pd.read_csv('/content/Historical_Adjusted_Close.csv', index_col=0)

S_hist.dropna(axis=1, how='any', inplace=True)
S_hist.index = pd.to_datetime(S_hist.index)
print(S_hist.columns)

portfolio = list(S_hist.columns)
S_hist.tail()
```

```
Index(['AAPL', 'AXP', 'BA', 'CAT', 'CSCO', 'CVX', 'DIS', 'GE', 'GS', 'HD',
       'IBM', 'INTC', 'JNJ', 'JPM', 'KO', 'MCD', 'MMM', 'MRK', 'MSFT', 'NKE',
       'PFE', 'PG', 'T', 'TRV', 'UNH', 'UTX', 'VZ', 'WMT', 'XOM'],
      dtype='object')
```

Out[24]:

	AAPL	AXP	BA	CAT	CSCO	CVX	DIS	GE	GS	HD	...	NKE	PFE	PG	
2018-05-30	187.5000	98.99	358.19	155.46	42.85	125.16	99.9800	14.17	229.16	187.09	...	72.0320	36.050	74.8900	3
2018-05-31	186.8700	98.30	352.16	151.91	42.71	124.30	99.4700	14.08	225.88	186.55	...	71.6032	35.930	73.1700	3
2018-06-01	190.2400	98.25	356.72	153.52	43.66	123.85	99.3600	14.10	228.35	187.35	...	72.7600	36.250	73.4500	3
2018-06-04	191.8300	99.45	360.73	152.50	43.58	122.26	100.2400	13.71	229.89	191.36	...	73.8300	36.450	74.1800	3
2018-06-05	192.8861	99.05	360.00	152.66	43.58	122.88	100.1144	13.85	228.90	191.05	...	74.1168	36.485	73.9755	3

5 rows × 29 columns

Select a Subperiod of the Historical Data

In [31]:

```
nYears = 1.5
start = S_hist.index[-int(nYears*252)]
end = S_hist.index[-1]

print('Start Date:', start)
print('  End Date:', end)

S = S_hist.loc[start:end]
S.head()
```

```
Start Date: 2016-12-02 00:00:00
  End Date: 2018-06-05 00:00:00
```

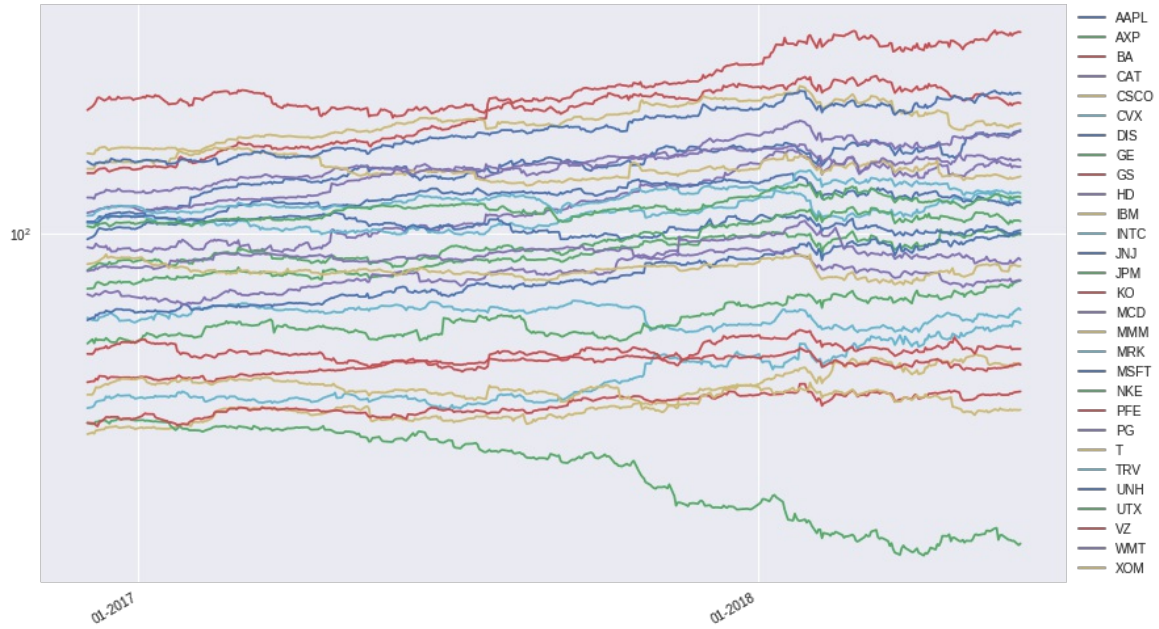
Out[31]:

	AAPL	AXP	BA	CAT	CSCO	CVX	DIS	GE	GS	HD	...	NKE	
2016-12-02	107.3034	70.2217	146.5851	91.4881	27.8083	106.6567	96.2544	29.8466	219.1815	125.5267	...	49.5074	29.
2016-12-05	106.5321	70.3879	146.4985	90.8246	28.0745	106.8926	97.6811	29.6276	224.2744	125.3430	...	50.8712	29.
2016-12-06	107.3522	70.6615	146.5755	91.5651	27.8844	106.4301	98.3651	29.6847	227.0515	124.4345	...	49.6154	29.
2016-12-07	108.4067	72.5963	148.4048	93.5941	28.4738	108.0158	99.6648	30.0942	231.1533	128.0300	...	51.1165	29.
2016-12-08	109.4709	73.2022	149.6083	92.5747	28.4738	108.7048	101.7853	30.0276	236.9331	128.0977	...	50.5769	29.

5 rows × 29 columns

In [33]:

```
fig, ax = plt.subplots(figsize=(14,9))
S.plot(ax=ax, logy=True)

ax.xaxis.set_major_locator(mdates.YearLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
plt.legend(loc='center left', bbox_to_anchor=(1.0, 0.5));
```



Return on a Portfolio

Given a portfolio with value W_t at time t, return on the portfolio at $t_{t+\delta t}$ is defined as

$$r_{t+\delta t} = \frac{W_{t+\delta t} - W_t}{W_t}$$

For the period from $[t, t+\delta t)$ we assume there are $n_{j,t}$ shares of asset j with a starting value of $S_{j,t}$ per share. The initial and final values of the portfolio are then

$$\begin{aligned} W_t &= \sum_{j=1}^{J} n_{j,t} S_{j,t} \\ W_{t+\delta t} &= \sum_{j=1}^{J} n_{j,t} S_{j,t+\delta t} \end{aligned}$$

The return of the portfolio is given by

$$\begin{aligned} r_{t+\delta t} &= \frac{W_{t+\delta t} - W_t}{W_t} \\ &= \frac{\sum_{j=1}^{J} n_{j,t} S_{j,t+\delta t} - \sum_{j=1}^{J} n_{j,t} S_{j,t}}{W_t} \\ &= \frac{\sum_{j=1}^{J} n_{j,t} S_{j,t} r_{j,t+\delta t}}{W_t} \\ &= \sum_{j=1}^{J} \frac{n_{j,t} S_{j,t}}{W_t} r_{j,t+\delta t} \end{aligned}$$

where $r_{j,t+\delta t}$ is the return on asset j at time $t+\delta t$.

Defining $W_{j,t} = n_{j,t} S_{j,t}$ as the wealth invested in asset j at time t, then $w_{j,t} = n_{j,t} S_{j,t} / W_t$ is the fraction of total wealth invested in asset j at time t. The portfolio return is then given by

$$r_{t+\delta t} = \sum_{j=1}^{J} w_{j,t} r_{j,t+\delta t}$$

on a single interval extending from t to $t+\delta t$.

Equally Weighted Portfolio

An equally weighted portfolio allocates an equal amount of capital to each component of the portfolio. The allocation can be done once and held fixed thereafter, or could be reallocated periodically as asset prices change in relation to one another.

Constant Fixed Allocation

If the initial allocation among J assets takes place at $t = 0$, then

$$w_{j,0} = \frac{1}{J} = \frac{n_{j,0} S_{j,t=0}}{W_0}$$

The number of assets of type j included in the portfolio is given by

$$n_{j,0} = \frac{W_0}{J S_{j,0}}$$

which is then fixed for all later times $t > 0$. The value of the portfolio is given by

$$\begin{aligned} W_t &= \sum_{j=1}^{J} n_{j,0} S_{j,t} \\ &= \frac{W_0}{J} \sum_{j=1}^{J} \frac{S_{j,t}}{S_{j,0}} \end{aligned}$$

Note that this portfolio is guaranteed to be equally weighted only at $t = 0$. Changes in the relative prices of assets cause the relative weights of assets in the portfolio to change over time.

Continually Rebalanced

Maitaining an equally weighted portfolio requires buying and selling of component assests as prices change relative to each other. To maintain an equally portfolio comprised of J assets where the weights are constant in time,

$$w_{j,t} = \frac{1}{J} = \frac{n_{j,t} S_{j,t}}{W_t} \qquad \forall j, t$$

Assuming the rebalancing occurs at fixed points in time t_k separated by time steps δt, then on each half-closed interval $[t_k, t_k + \delta t)$

$$n_{j,t} = \frac{W_{t_k}}{J S_{j,t_k}}$$

The portfolio

$$\begin{aligned} W_{t_k + \delta t} &= \sum_{j=1}^{J} n_{j,t_k} S_{j,t_k + \delta t} \\ W_{t_k + \delta t} &= W_{t_k} \sum_{j=1}^{J} \frac{S_{j,t_k + \delta t}}{J S_{j,t_k}} \end{aligned}$$

Letting $t_{k+1} = t_k + \delta t$, then the following recursion describes the dynamics of an equally weighted, continually rebalanced portfolio at the time steps $t_0, t_1, \ldots$. Starting with values W_{t_0} and S_{j,t_0} ,

$$\begin{aligned} n_{j,t_k} &= \frac{W_{t_k}}{J S_{j,t_k}} \\ W_{t_{k+1}} &= \sum_{j=1}^{J} n_{j,t_k} S_{j,t_{k+1}} \end{aligned}$$

which can be simulated as a single equation

$$W_{t_{k+1}} = W_{t_k} \sum_{j=1}^{J} \frac{S_{j,t_{k+1}}}{J S_{j,t_k}}$$

or in closed-form

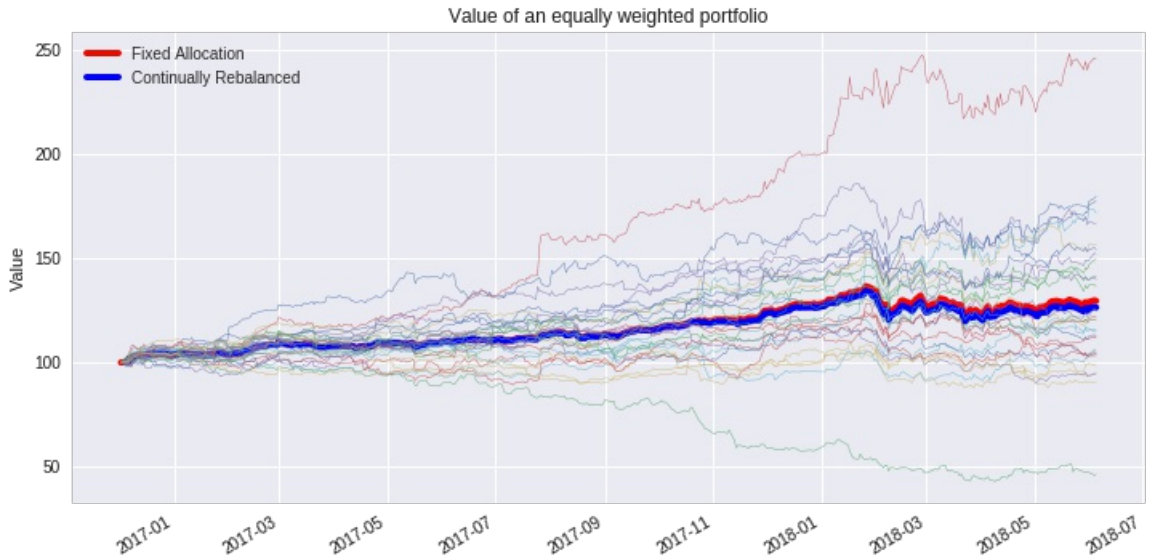
$$W_{t_K} = W_0 \prod_{k=0}^{K-1} \sum_{j=1}^{J} \frac{S_{j,t_{k+1}}}{J S_{j,t_k}}$$

In [49]:

```
plt.figure(figsize=(12,6))

portfolio = S.columns
J = len(portfolio)

# equal weight with no rebalancing
n = 100.0/S.iloc[0]/J
W_fixed = sum(n[s]*S[s] for s in portfolio)
W_fixed.plot(color='r',lw=4)

# equal weighting with continual rebalancing
#W_rebal = [100]
#for k in range(1, len(S.index)):
#    n = W_rebal[k-1]/S.iloc[k-1,:]/len(portfolio)
#    W_rebal.append(sum(n[j]*S.iloc[k,j] for j in range(0, len(portfolio))))

#W_rebal = pd.Series(W_rebal, index=S.index)
#W_rebal.plot(lw=4, color='b')

# equal weighting with continual rebalancing
R = (S[1:]/S.shift(1)[1:]).sum(axis=1)/len(portfolio)
W_rebal = 100*R.cumprod()
W_rebal.plot(color='b', lw=4)

# individual assets
for s in portfolio:
    (100.0*S[s]/S[s][0]).plot(lw=0.4)

plt.legend(['Fixed Allocation','Continually Rebalanced'])

plt.ylabel('Value');
plt.title('Value of an equally weighted portfolio')
```

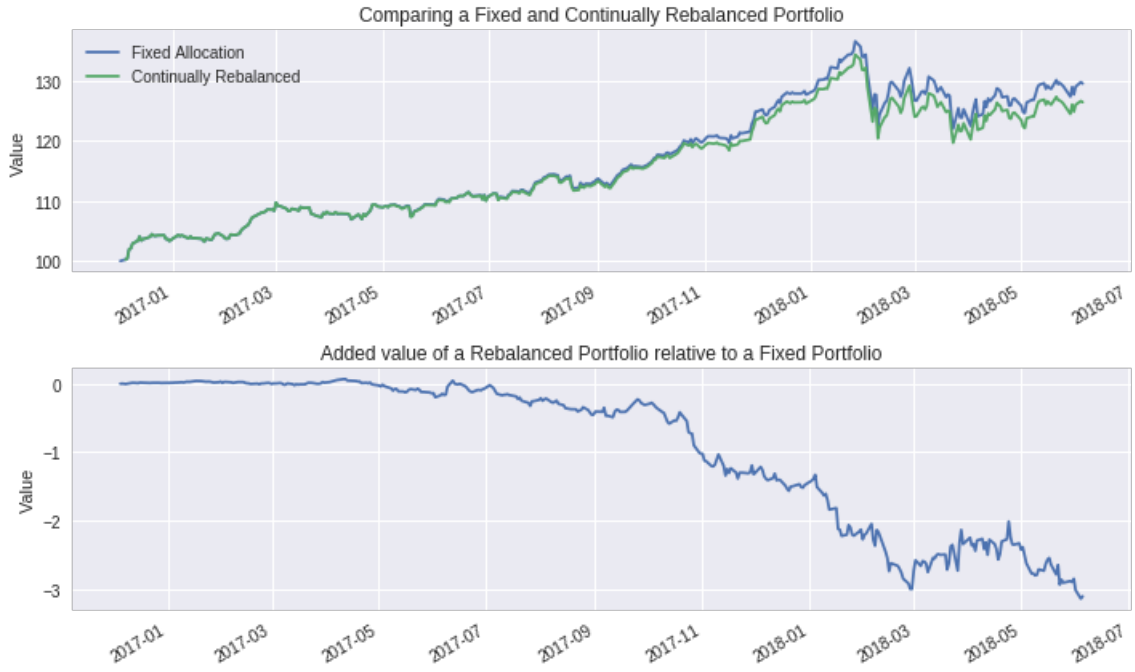
Out[49]:

```
Text(0.5,1,'Value of an equally weighted portfolio')
```



In [50]:

```
plt.figure(figsize=(10,6))

plt.subplot(2,1,1)
W_fixed.plot()
W_rebal.plot()
plt.legend(['Fixed Allocation','Continually Rebalanced'])
plt.ylabel('Value')
plt.title('Comparing a Fixed and Continually Rebalanced Portfolio')

plt.subplot(2,1,2)
(W_rebal-W_fixed).plot()
plt.title('Added value of a Rebalanced Portfolio relative to a Fixed Portfolio')
plt.ylabel('Value')

plt.tight_layout()
```



Component Returns

Given data on the prices for a set of assets over an historical period $t_0, t_1, \ldots, t_K$, an estimate the mean arithmetic return is given by the mean value

$$\begin{aligned}\hat{r}_{j,t_K} &= \frac{1}{K}\sum_{k=1}^{K} r_{t_k} \\ &= \sum_{k=1}^{K} \frac{S_{j,t_k} - S_{j,t_{k-1}}}{S_{j,t_{k-1}}}\end{aligned}$$

At any point in time, t_k, a mean return can be computed using the previous H intervals

$$\begin{aligned}\hat{r}^{H}_{j,t_k} &= \frac{1}{H}\sum_{h=0}^{H-1} r_{t_{k-h}} \\ &= \frac{1}{H}\sum_{h=0}^{H-1} \frac{S_{j,t_{k-h}} - S_{j,t_{k-h-1}}}{S_{j,t_{k-h-1}}}\end{aligned}$$

Arithmetic returns are computed so that subsequent calculations combine returns across components of a portfolio.

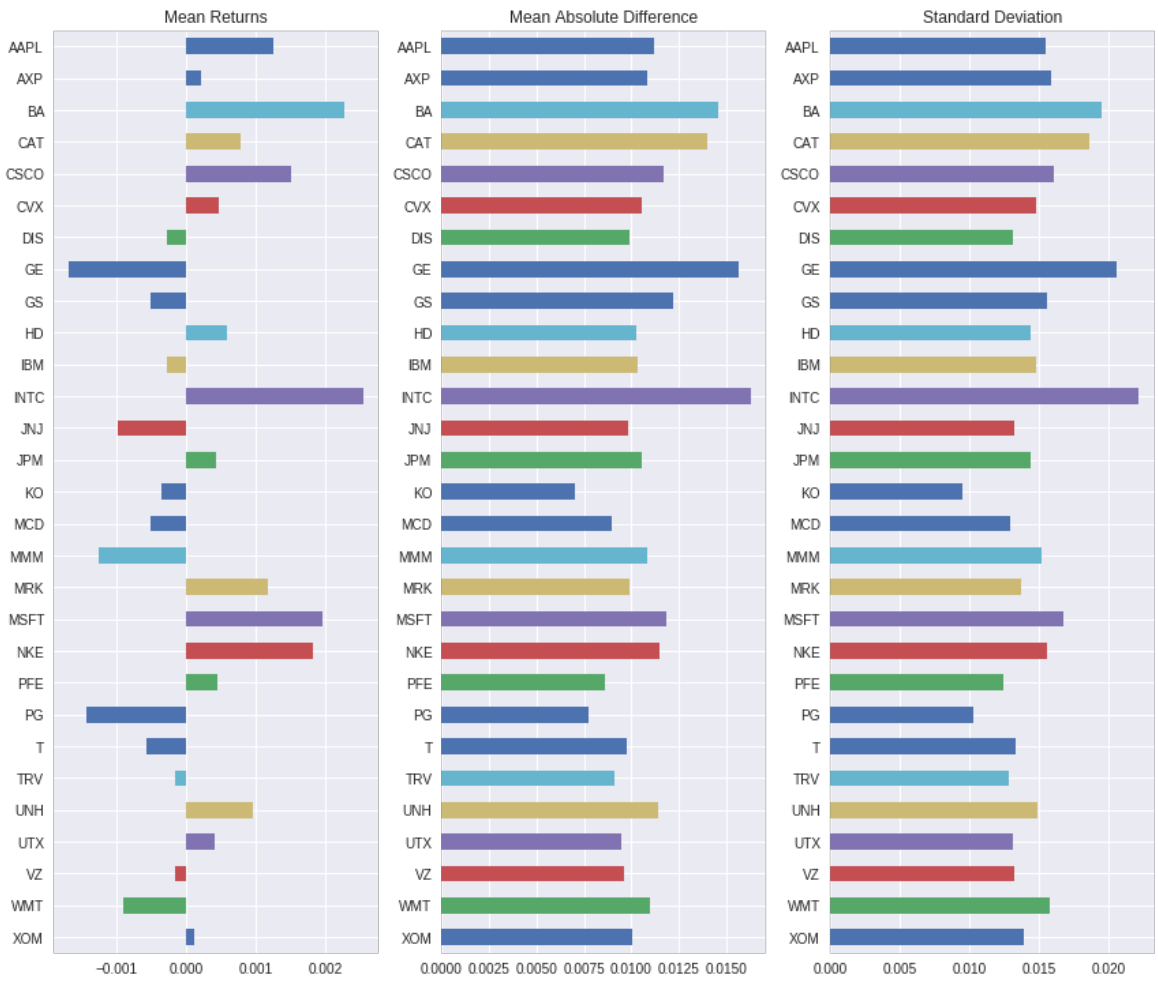
Measuring Deviation in Component Returns

Standard Deviation

Mean Absolute Deviation

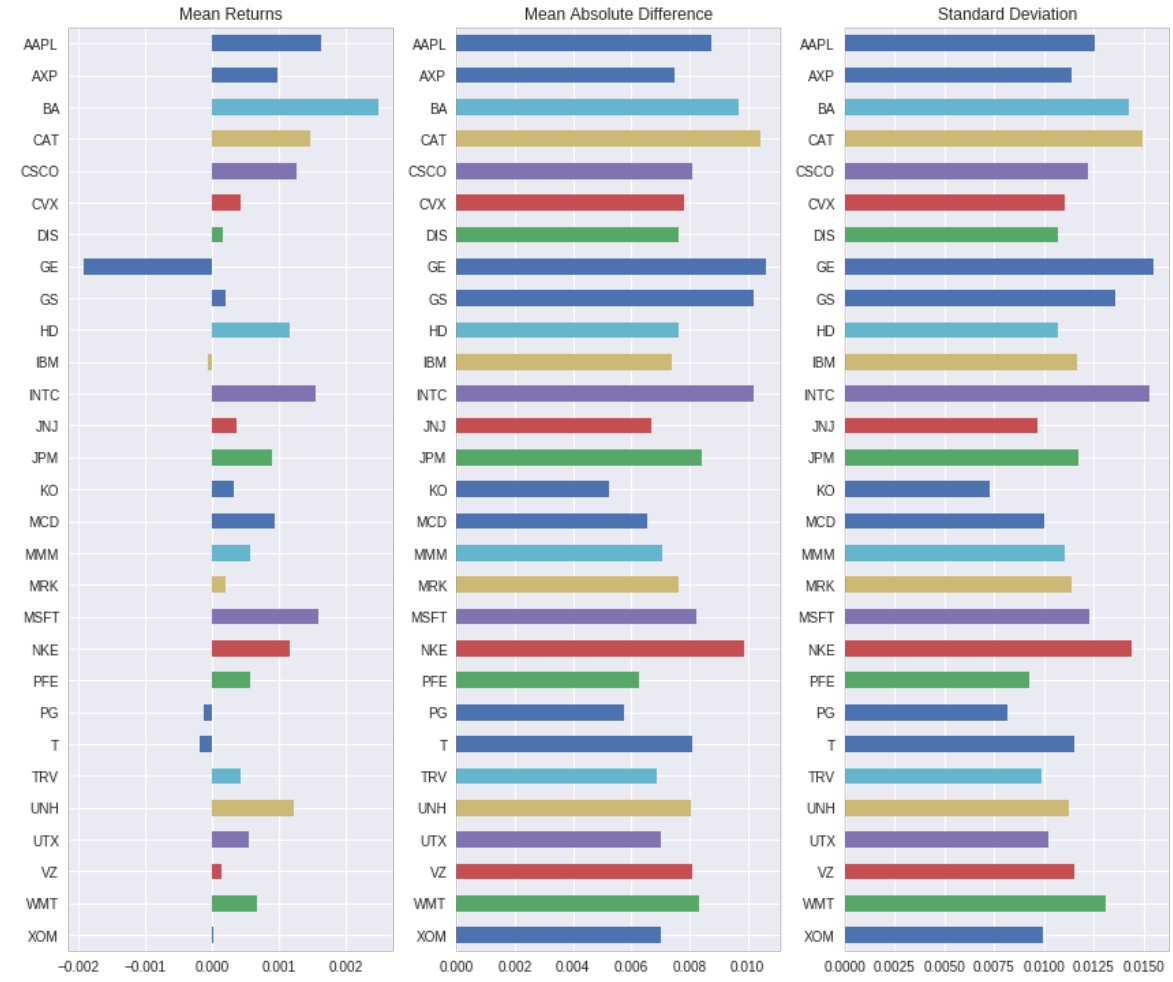
In [51]:

```
def roll(H):
    k = len(S.index)
    R = S[k-H-1:k].diff()[1:]/S[k-H-1:k].shift(1)[1:]
    AD = abs(R - R.mean())

    plt.figure(figsize = (12, 0.35*len(R.columns)))
    idx = R.columns.argsort()[::-1]

    plt.subplot(1,3,1)
    R.mean().iloc[idx].plot(kind='barh')
    plt.title('Mean Returns');

    plt.subplot(1,3,2)
    AD.mean().iloc[idx].plot(kind='barh')
    plt.title('Mean Absolute Difference')

    plt.subplot(1,3,3)
    R.std().iloc[idx].plot(kind='barh')
    plt.title('Standard Deviation')

    plt.tight_layout()

roll(500)
```


Mean Returns
Mean Absolute Difference
Standard Deviation
AAPL
AXP
BA
CAT
CSCO
CVX
DIS
GE
GS
HD
IBM
INTC
JNJ
JPM
KO
MCD
MMM
MRK
MSFT
NKE
PFE
PG
T
TRV
UNH
UTX
VZ
WMT
XOM
−0.001
0.000
0.001
0.002
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.000
0.005
0.010
0.015
0.020

In [52]:

```
R = (S.diff()[1:]/S.shift(1)[1:]).dropna(axis=0, how='all')
AD = abs(R - R.mean())

plt.figure(figsize = (12, 0.35*len(R.columns)))
idx = R.columns.argsort()[::-1]

plt.subplot(1,3,1)
R.mean().iloc[idx].plot(kind='barh')
plt.title('Mean Returns');

plt.subplot(1,3,2)
AD.mean().iloc[idx].plot(kind='barh')
plt.title('Mean Absolute Difference')

plt.subplot(1,3,3)
R.std().iloc[idx].plot(kind='barh')
plt.title('Standard Deviation')

plt.tight_layout()
```

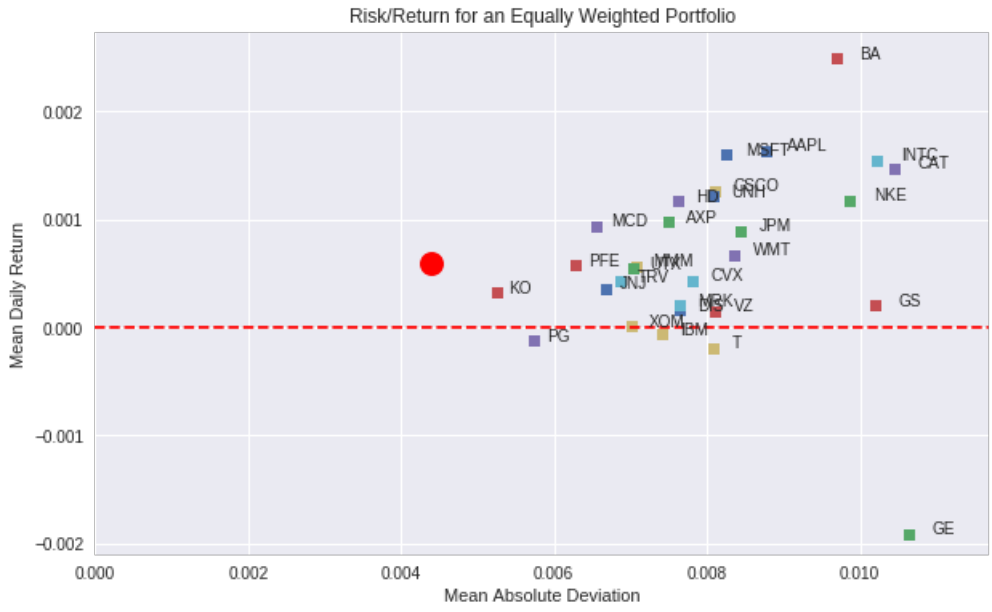


In [53]:

```
plt.plot(R.std(), AD.mean(), 'o')
plt.xlabel('Standard Deviation')
plt.ylabel('Mean Absolute Deviation')

plt.plot([0,R.std().max()],[0,np.sqrt(2.0/np.pi)*R.std().max()])
plt.legend(['Portfolio Components','sqrt(2/np.pi)'],loc='best');
```

In [69]:

```
plt.figure(figsize=(10,6))
for s in portfolio:
    plt.plot(AD[s].mean(), R[s].mean(),'s')
    plt.text(AD[s].mean()*1.03, R[s].mean(), s)

R_equal = W_rebal.diff()[1:]/W_rebal[1:]
#R_equal = np.log(W_rebal/W_rebal.shift(+1))
M_equal = abs(R_equal-R_equal.mean()).mean()

plt.plot(M_equal, R_equal.mean(), 'ro', ms=15)

plt.xlim(0, 1.1*max(AD.mean()))
plt.ylim(min(0, 1.1*min(R.mean())), 1.1*max(R.mean()))
plt.plot(plt.xlim(),[0,0],'r--');
plt.title('Risk/Return for an Equally Weighted Portfolio')
plt.xlabel('Mean Absolute Deviation')
plt.ylabel('Mean Daily Return');
```



MAD Porfolio

The linear program is formulated and solved using the pulp package.

In [70]:

```
R.head()
```

Out[70]:

	AAPL	AXP	BA	CAT	CSCO	CVX	DIS	GE	GS	HD	...	NK
2016-12-05	-0.007188	0.002367	-0.000591	-0.007252	0.009573	0.002212	0.014822	-0.007338	0.023236	-0.001463	...	0.02754
2016-12-06	0.007698	0.003887	0.000526	0.008153	-0.006771	-0.004327	0.007002	0.001927	0.012383	-0.007248	...	0.02468
2016-12-07	0.009823	0.027381	0.012480	0.022159	0.021137	0.014899	0.013213	0.013795	0.018066	0.028895	...	0.03025
2016-12-08	0.009817	0.008346	0.008110	-0.010892	0.000000	0.006379	0.021276	-0.002213	0.025004	0.000529	...	0.01055
2016-12-09	0.016322	-0.001869	0.007079	-0.007686	0.003674	0.005557	0.014316	0.007929	0.001657	0.006489	...	0.00329

5 rows × 29 columns

The decision variables will be indexed by date/time. The pandas dataframes containing the returns data are indexed by timestamps that include characters that cannot be used by the GLPK solver. Therefore we create a dictionary to translate the pandas timestamps to strings that can be read as members of a GLPK set. The strings denote seconds in the current epoch as defined by python.

In [71]:

```
a = R - R.mean()
a.head()
```

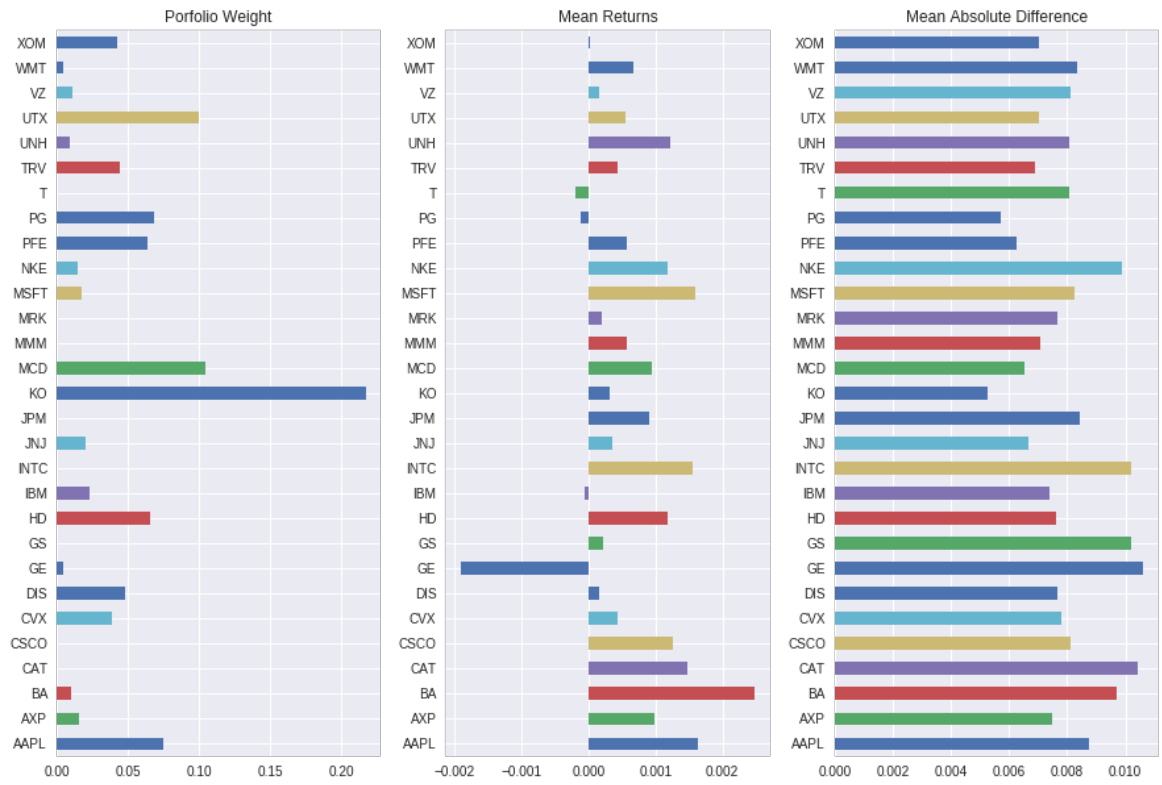
Out[71]:

	AAPL	AXP	BA	CAT	CSCO	CVX	DIS	GE	GS	HD	...	NK
2016-12-05	-0.008823	0.001389	-0.003078	-0.008722	0.008305	0.001775	0.014661	-0.005425	0.023029	-0.002636	...	0.02637
2016-12-06	0.006063	0.002910	-0.001961	0.006683	-0.008038	-0.004763	0.006841	0.003840	0.012176	-0.008420	...	0.02585
2016-12-07	0.008188	0.026404	0.009993	0.020689	0.019870	0.014462	0.013051	0.015708	0.017858	0.027722	...	0.02908
2016-12-08	0.008182	0.007369	0.005623	-0.012362	-0.001267	0.005942	0.021115	-0.000300	0.024797	-0.000643	...	0.01173
2016-12-09	0.014687	-0.002846	0.004592	-0.009156	0.002406	0.005121	0.014155	0.009842	0.001450	0.005317	...	0.00212

5 rows × 29 columns

In [72]:

```
from pyomo.environ import *

a = R - R.mean()

m = ConcreteModel()

m.w = Var(R.columns, domain=NonNegativeReals)
m.y = Var(R.index, domain=NonNegativeReals)

m.MAD = Objective(expr=sum(m.y[t] for t in R.index)/len(R.index), sense=minimize)

m.c1 = Constraint(R.index, rule = lambda m, t:
    m.y[t] + sum(a.loc[t,s]*m.w[s] for s in R.columns) >= 0)

m.c2 = Constraint(R.index, rule = lambda m, t:
    m.y[t] - sum(a.loc[t,s]*m.w[s] for s in R.columns) >= 0)

m.c3 = Constraint(expr=sum(R[s].mean()*m.w[s] for s in R.columns) >= R_equal.mean())

m.c4 = Constraint(expr=sum(m.w[s] for s in R.columns)==1)

SolverFactory('glpk').solve(m)

w = {s: m.w[s]() for s in R.columns}

plt.figure(figsize = (15,0.35*len(R.columns)))

plt.subplot(1,3,1)
pd.Series(w).plot(kind='barh')
plt.title('Porfolio Weight');

plt.subplot(1,3,2)
R.mean().plot(kind='barh')
plt.title('Mean Returns');

plt.subplot(1,3,3)
AD.mean().plot(kind='barh')
plt.title('Mean Absolute Difference');
```



In [76]:

```
P_mad = pd.Series(0,index=S.index)
for s in portfolio:
    P_mad += 100.0*w[s]*S[s]/S[s][0]

plt.figure(figsize=(12,6))
P_mad.plot()
W_rebal.plot()
plt.legend(['MAD','Equal'],loc='best')
plt.ylabel('Unit Value')
```

Out[76]:

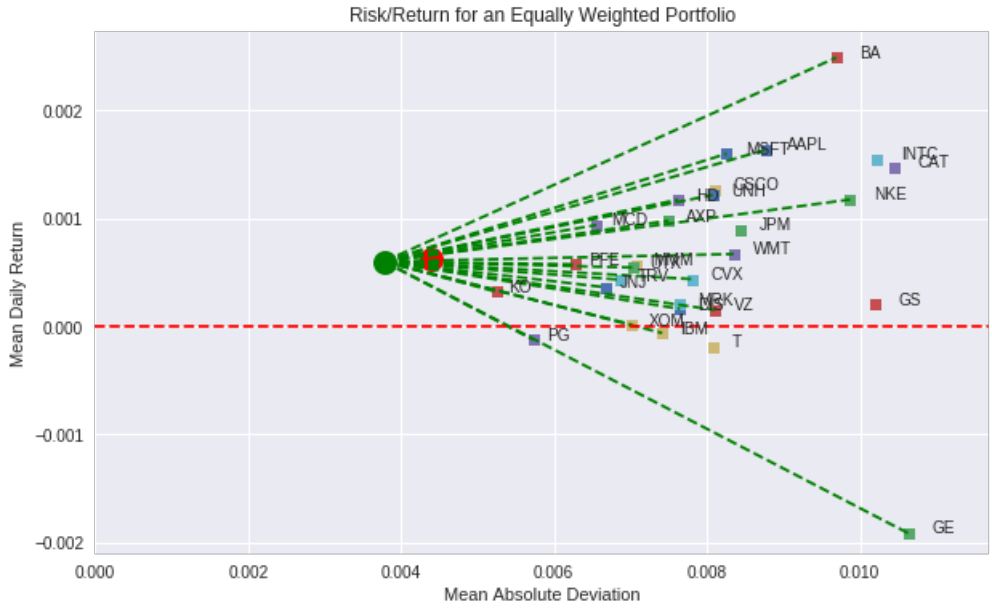
```
Text(0,0.5,'Unit Value')
```

In [80]:

```
plt.figure(figsize=(10,6))
for s in portfolio:
    plt.plot(AD[s].mean(), R[s].mean(),'s')
    plt.text(AD[s].mean()*1.03, R[s].mean(), s)

#R_equal = P_equal.diff()[1:]/P_equal[1:]
R_equal = np.log(W_rebal/W_rebal.shift(+1))
M_equal = abs(R_equal-R_equal.mean()).mean()

plt.plot(M_equal, R_equal.mean(), 'ro', ms=15)

#R_mad = P_mad.diff()[1:]/P_mad[1:]
R_mad = np.log(P_mad/P_mad.shift(+1))
M_mad = abs(R_mad-R_mad.mean()).mean()

plt.plot(M_mad, R_mad.mean(), 'go', ms=15)

for s in portfolio:
    if w[s] >= 0.0001:
        plt.plot([M_mad, AD[s].mean()],[R_mad.mean(), R[s].mean()],'g--')
    if w[s] <= -0.0001:
        plt.plot([M_mad, AD[s].mean()],[R_mad.mean(), R[s].mean()],'r--')

plt.xlim(0, 1.1*max(AD.mean()))
plt.ylim(min(0, 1.1*min(R.mean())), 1.1*max(R.mean()))
plt.plot(plt.xlim(),[0,0],'r--');
plt.title('Risk/Return for an Equally Weighted Portfolio')
plt.xlabel('Mean Absolute Deviation')
plt.ylabel('Mean Daily Return')
```

Out[80]:

```
Text(0,0.5,'Mean Daily Return')
```



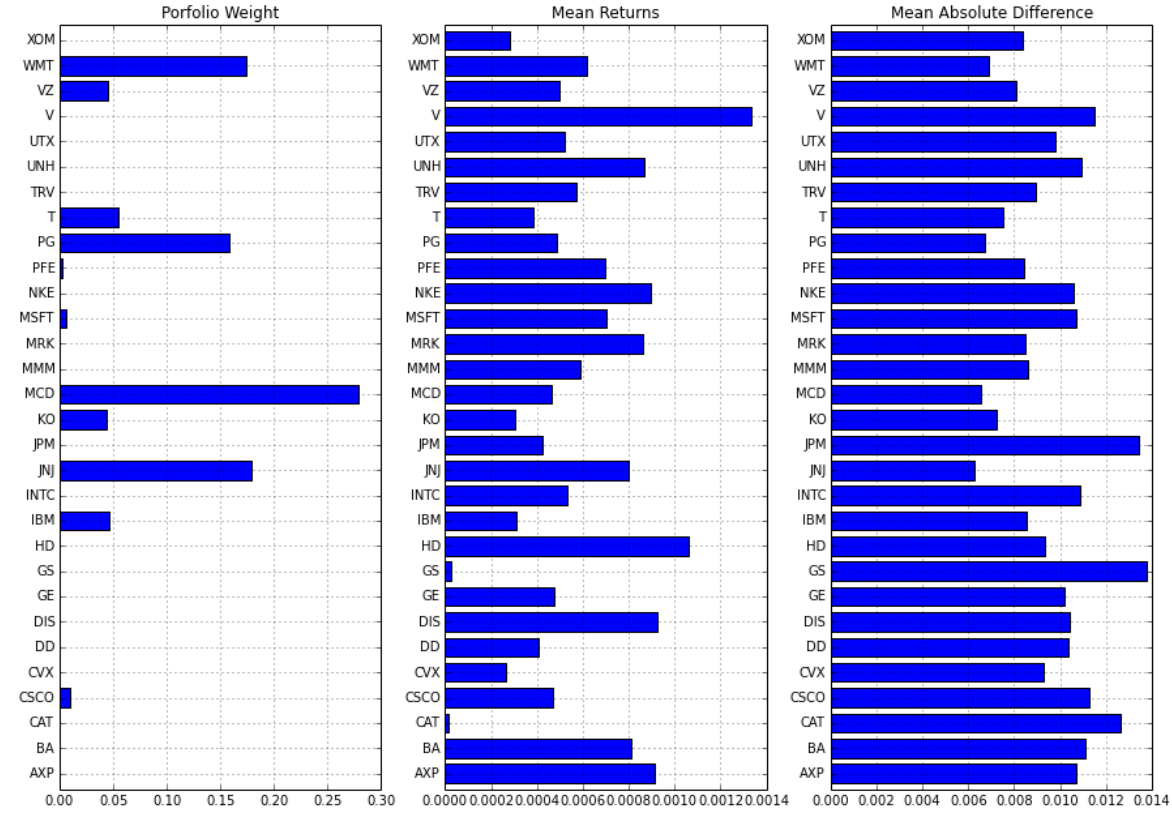
In [ ]:

```
import pulp

# mean absolute deviation for the portfolio
m = pulp.LpVariable('m', lowBound = 0)

# dictionary of portfolio weights
w = pulp.LpVariable.dicts('w', portfolio, lowBound = 0)

# dictionary of absolute deviations of portfolio returns
y = pulp.LpVariable.dicts('y', t.values(), lowBound = 0)
z = pulp.LpVariable.dicts('z', t.values(), lowBound = 0)

# create problem instance
lp = pulp.LpProblem('MAD Portfolio',pulp.LpMinimize)

# add objective
lp += m

# calculate mean absolute deviation of portfolio returns
lp += m == pulp.lpSum([(y[k] + z[k]) for k in t.values()])/float(len(t))

# relate the absolute deviations to deviations in the portfolio returns
for ts in returns.index:
    lp += y[t[ts]] - z[t[ts]] == pulp.lpSum([w[s]*(returns[s][ts]-returns[s].mean()) for
s in portfolio])

# portfolio weights
lp += pulp.lpSum([w[s] for s in portfolio]) == 1.0

# bound on average portfolio return
lp += pulp.lpSum([w[s]*(returns[s].mean()) for s in portfolio]) >= 0*R_equal.mean()

lp.solve()
print pulp.LpStatus[lp.status]
```

```
  File "<ipython-input-164-77bf056feb2a>", line 35
    print pulp.LpStatus[lp.status]
                ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean
print(pulp.LpStatus[lp.status])?
```

In [ ]:

```
figure(figsize = (15,0.35*len(returns.columns)))

ws = pd.Series({s: w[s].varValue for s in portfolio},index=portfolio)

subplot(1,3,1)
ws.plot(kind='barh')
title('Porfolio Weight');

subplot(1,3,2)
returns.mean().plot(kind='barh')
title('Mean Returns');

subplot(1,3,3)
abs(returns-returns.mean()).mean().plot(kind='barh')
title('Mean Absolute Difference');
```

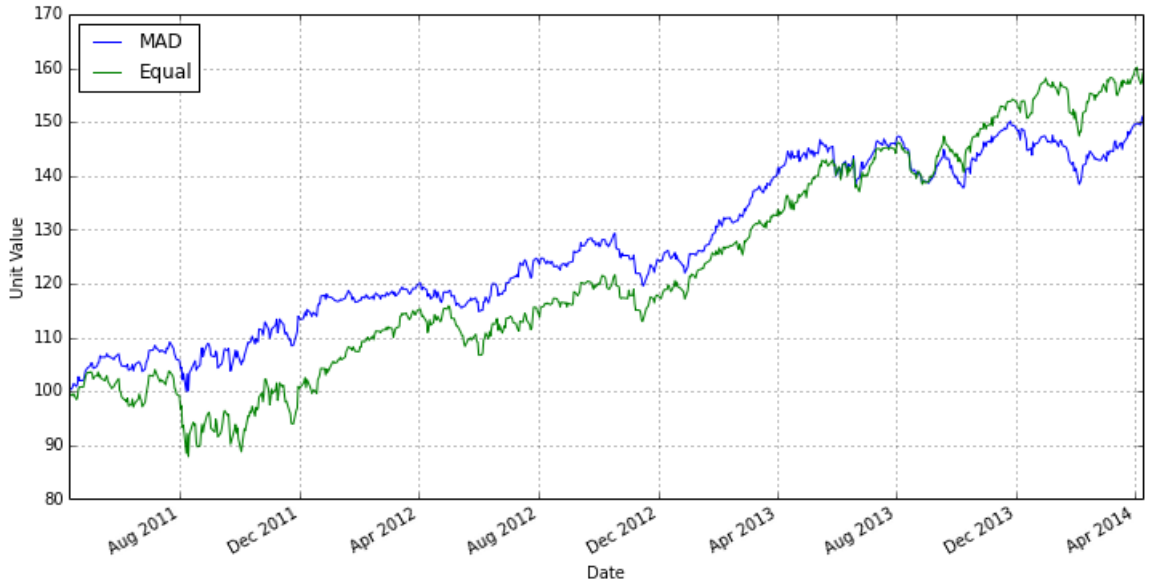


In [ ]:

```
P_mad = pd.Series(0,index=adjclose.index)
for s in portfolio:
    P_mad += 100.0*ws[s]*adjclose[s]/adjclose[s][0]

figure(figsize=(12,6))
P_mad.plot()
P_equal.plot()
legend(['MAD','Equal'],loc='best')
ylabel('Unit Value')
```

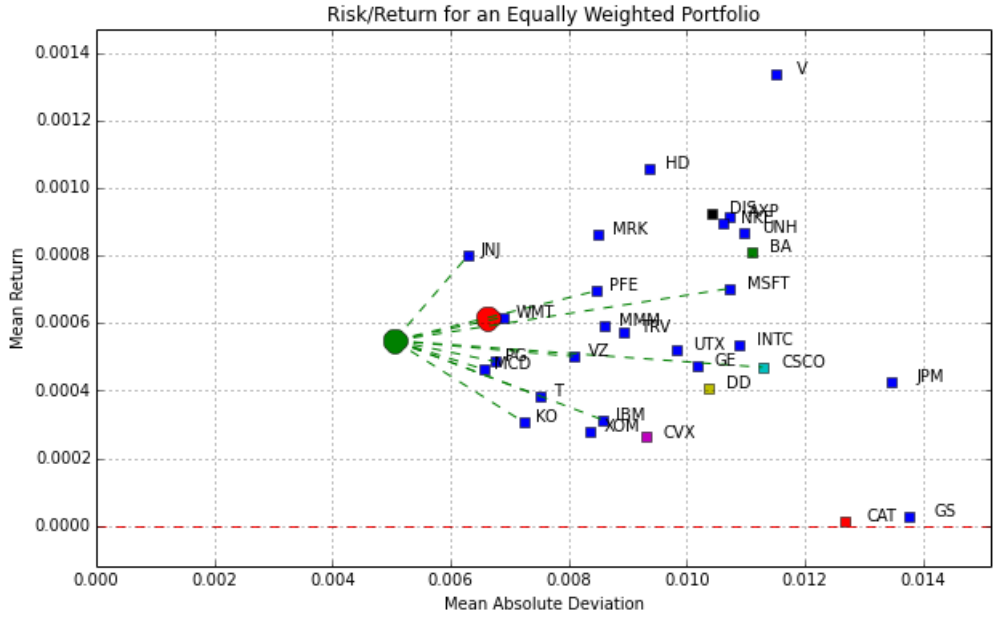
Out[ ]:

```
<matplotlib.text.Text at 0x10e0bed90>
```



In [ ]:

```
figure(figsize=(10,6))
for s in portfolio:
    plot(mad[s],rmean[s],'s')
    text(mad[s]*1.03,rmean[s],s)

axis([0, 1.1*max(mad), min([0,min(rmean)-.1*(max(rmean)-min(rmean))]), 1.1*max(rmean)])
ax = axis()
plot([ax[0],ax[1]],[0,0],'r--');

#R_equal = P_equal.diff()[1:]/P_equal[1:]
R_equal = log(P_equal/P_equal.shift(+1))
M_equal = abs(R_equal-R_equal.mean()).mean()

plot(M_equal,R_equal.mean(),'ro',ms=15)

#R_mad = P_mad.diff()[1:]/P_mad[1:]
R_mad = log(P_mad/P_mad.shift(+1))
M_mad = abs(R_mad-R_mad.mean()).mean()

for s in portfolio:
    if ws[s] >= 0.0001:
        plot([M_mad,mad[s]],[R_mad.mean(),rmean[s]],'g--')

plot(M_mad,R_mad.mean(),'go',ms=15)

title('Risk/Return for an Equally Weighted Portfolio')
xlabel('Mean Absolute Deviation')
ylabel('Mean Return')

grid();
```



Problem 1: Solve for Dominating MAD Portfolio

In [ ]:

```
lp = pulp.LpProblem('MAD Portfolio',pulp.LpMinimize)
lp += m
lp += m == pulp.lpSum([(y[t] + z[t])/float(len(returns.index)) for t in tmap.values()])
lp += pulp.lpSum([w[s] for s in symbols]) == 1.0

for t in returns.index:
    lp += y[tsec[t]] - z[tsec[t]] == pulp.lpSum([w[s]*(returns[s][t]-rmean[s]) for s in
symbols])

lp.solve()
m_min = m.varValue
m_min
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-94-7e6a74985eb9> in <module>()
      1 lp = pulp.LpProblem('MAD Portfolio',pulp.LpMinimize)
      2 lp += m
----> 3 lp += m == pulp.lpSum([(y[t] + z[t])/float(len(returns.index)) for t in tmap.val
ues()])
      4 lp += pulp.lpSum([w[s] for s in symbols]) == 1.0
      5

NameError: name 'tmap' is not defined
```

In [ ]:

```
# Solve for maximum return at minimum MAD

r = pulp.LpVariable('r',lowBound = 0)

lp = pulp.LpProblem('MAD Portfolio',pulp.LpMaximize)
lp += r
lp += r == pulp.lpSum([w[s]*rmean[s] for s in symbols])
lp += m_min == pulp.lpSum([(y[t] + z[t])/float(len(returns.index)) for t in tmap.values
()])
lp += pulp.lpSum([w[s] for s in symbols]) == 1.0
for t in returns.index:
    lp += y[tsec[t]] - z[tsec[t]] == pulp.lpSum([w[s]*(returns[s][t]-rmean[s]) for s in
symbols])

lp.solve()
r_min = r.varValue
w_min = pd.Series([pulp.value(w[s]) for s in symbols], index= symbols)
```

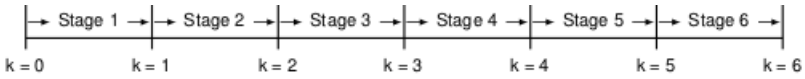
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-95-bb19d9078a1c> in <module>()
      5 lp = pulp.LpProblem('MAD Portfolio',pulp.LpMaximize)
      6 lp += r
----> 7 lp += r == pulp.lpSum([w[s]*rmean[s] for s in symbols])
      8 lp += m_min == pulp.lpSum([(y[t] + z[t])/float(len(returns.index)) for t in tmap
.values()])
      9 lp += pulp.lpSum([w[s] for s in symbols]) == 1.0

NameError: name 'symbols' is not defined
```

8.5 Real Options

Implementing a Replicating Portfolio in Pyomo

A replicating portfolio is the key concept for using of a binomial model to price options. In a nutshell, the value of an option is the money needed to construct a portfolio that replicates the option's payoff for every possible outcome. The replicating portfolio can be extended to applications involving 'real assets', including processes that convert commodity resources into higher value products.

The period of the contract is broken into N stages where $k = 0$ denotes the start of the first stage of the option contract and $k = N$ denotes the end of the last stage.



Time Line for a Replicating Portfolio

The value of the replicating portfolio at the start of stage $k+1$ in state s is given by

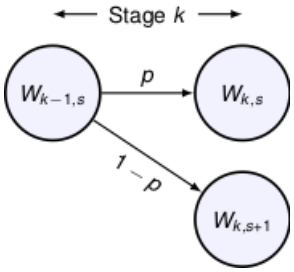
$$W_{k,s} = x_{k,s} S^f_{k,s} + y_{k,s}$$

The portfolio consists of $x_{k,s}$ units of an underlying asset and $y_{k,s}$ is in units of cash. The values of $x_{k,s}$ or $y_{k,s}$ can be positive or negative. $S^f_{k,s}$ is the price of the underlying asset which is subject to statistical variability. Cash is assumed to have an investment return r per period.

For each stage $k = 1, 2, \ldots, N$ and states $s = 0, 1, \ldots, k$, the value of the portfolio must

1. prepay expenses $P_{k,s}$ required to maintain the asset through period k,
2. pay out income $Q_{k,s}$ and $Q_{k,s+1}$ the asset generated in stage k,
3. finance the portfolio for the subsequent states,
4. be the greater than the value any option available at node (k,s).

The last proposition establishes the value of holding the option. For the binomial model, these requirements produce two constraints associated with each stage and state. For stage k



we write

$$\begin{aligned} x_{k-1,s} S^f_{k,s} + (y_{k-1,s} - P_{k,s})(1+r) &\geq W_{k,s} + Q_{k,s} \\ x_{k-1,s} S^f_{k,s+1} + (y_{k-1,s} - P_{k,s})(1+r) &\geq W_{k,s+1} + Q_{k,s+1} \end{aligned}$$

where we assume expenses are incurred immediately prior to the start of stage k, and income is accumulated immediately following completion of stage k.

Example 1. Value of a fixed cash flow

Suppose we are considering the purchase of lease on property that could generate $10,000 per year of income for the next 10 years. We assume there is virtually no risk that the full of those payments will not be received.

The value of the portfolio after stage k is designated W_k. The replicating portfolio consists of cash equivalents in the amount y_k that may be deposited with an annual interest rate r.

$$W_k = y_k$$

Upon expiration the value of the lease is zero, thus

$$W_N = y_N = 0$$

for a lease $N = 10$ years in length.

The rental income I_k is received at the beginning of each year k. The replicating portfolio must generate a payment equal to this amount at the beginning of the year, then at the end of the year retain enough value to self-finance the portfolio for the following year. This holds true for every year, thus

$$(1+r)(y_{k-1} - P_k) = y_k$$

Collectively, this establishes a difference equation

$$y_{k-1} = \frac{y_k}{1+r} + P_k$$

where $y_N = 0$. The solution for y_0 gives the initial amount of money needed to create a cash portfolio replicating the lease payments.

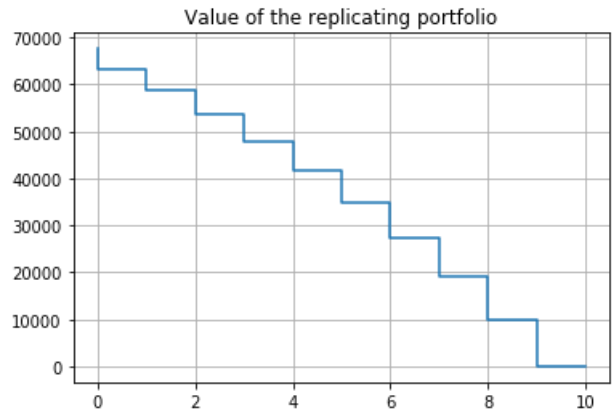
This model can be recast as an optimization problem by finding the minimum amount of initial cash is required to create a self-financing portfolio.

$$\min_{y_0, y_1, y_2, \ldots, y_N} \quad y_0$$

$$\begin{aligned}
&\text{subject to:} \\
&y_k && \geq 0 && \forall k = 0, 1, \ldots, N \\
&y_{k-1} && = \frac{y_k}{1+r} + P_k && \forall k = 1, 2, \ldots, N
\end{aligned}$$

In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from pyomo.environ import *

# problem data
N = 10
r = 0.10
P = 10000

# create model
m = ConcreteModel()

# decision variable
Stages = [k for k in range(0,N+1)]
m.y = Var(Stages,domain=NonNegativeReals)

# self-financing constraints
m.cons = ConstraintList()
for k in range(1,N+1):
    m.cons.add((m.y[k-1]-P)*(1+r) >= m.y[k])

# find minimum cost portfolio that satisfies the self-financing constraint
m.OBJ = Objective(expr = m.y[0], sense=minimize)

# solve
SolverFactory('glpk').solve(m)

plt.step(Stages,[m.y[k]() for k in Stages],where='pre')
plt.title('Value of the replicating portfolio')
plt.grid()
```



Example 2. Simplico Gold Mine

The Simplico Gold Mine is a well-known example due to Luenberger. Up to 10,000 ounces per year can be extracted from the gold mine at a processing cost of $200/oz. We assume the price of gold is currently $400/oz. and can modeled on a binomial lattice with u = 1.2, d=0.9, a probability 0.75 that the price increases. The cost of borrowing money is equal to 10% per year. Gold extracted during the year is sold at the end of the year for the price that prevailed at the start of the year.

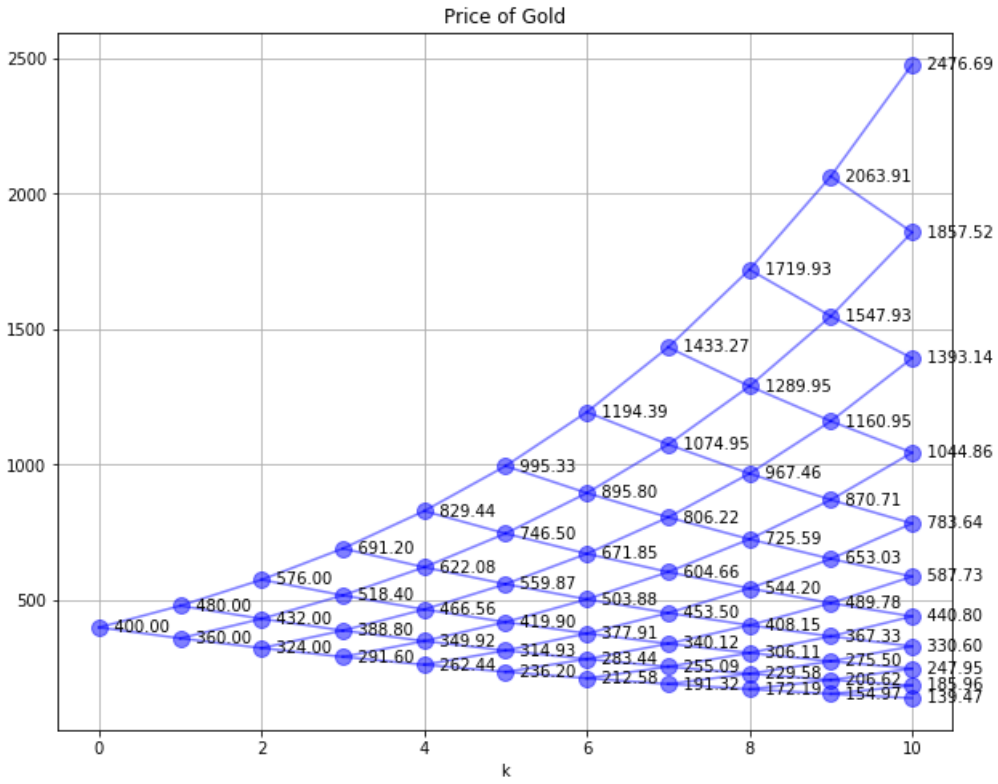
What is the value of a 10 year lease on the gold mine?

Solution

As a first step towards a solution, we compute a binomial lattice for the price of gold using the giving information.

In [2]:

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

# utility function to plot binomial tree
def SPdisplay(Sf,D,title=''):
    """
    SPdisplay(Sf,D,title)
        Sf: binomial tree of future prices
        D: binomial tree of data to display for each node
        title: plot title
    """
    plt.figure(figsize=(10,8))
    nPeriods = max([k for k,s in Sf.keys()]) + 1
    for k,s in Sf.keys():
        plt.plot(k,Sf[k,s],'.',ms=20,color='b',alpha=0.5)
        if (k > 0) & (s < k):
            plt.plot([k-1,k],[Sf[k-1,s],Sf[k,s]],'b',alpha=.5)
            plt.plot([k-1,k],[Sf[k-1,s],Sf[k,s+1]],'b',alpha=.5)
    for k,s in D.keys():
        plt.text(k,Sf[k,s],'   {0:.2f}'.format(D[k,s]),ha='left',va='center')
    plt.xlabel('k')
    plt.grid()
    plt.title(title)

N = 10
u = 1.2
d = 0.9
p = 0.75
r = 0.10

# initialize Sf
Sf = {}
Sf[0,0] = 400
for k in range(1,N+1):
    for s in range(0,k+1):
        Sf[k,s] = u**(k-s)*d**s*Sf[0,0]

SPdisplay(Sf,Sf,'Price of Gold')
```



The second step is to solve an optimization model for the lease value. Let $W_{k,s}$ denotethe value of the option at end of stage k and in state s. The terminal value of the lease is zero, i.e.,

$$W_{N,s} = 0 \qquad \forall s$$

The self-financing constraints are

$$\begin{aligned} x_{k-1,s}\, S^f_{k,s} + y_{k-1,s}(1+r) &\geq W_{k,s} + Q_{k,s} \\ x_{k-1,s}\, S^f_{k,s+1} + y_{k-1,s} - P_{k,s}(1+r) &\geq W_{k,s+1} + Q_{k,s+1} \end{aligned}$$

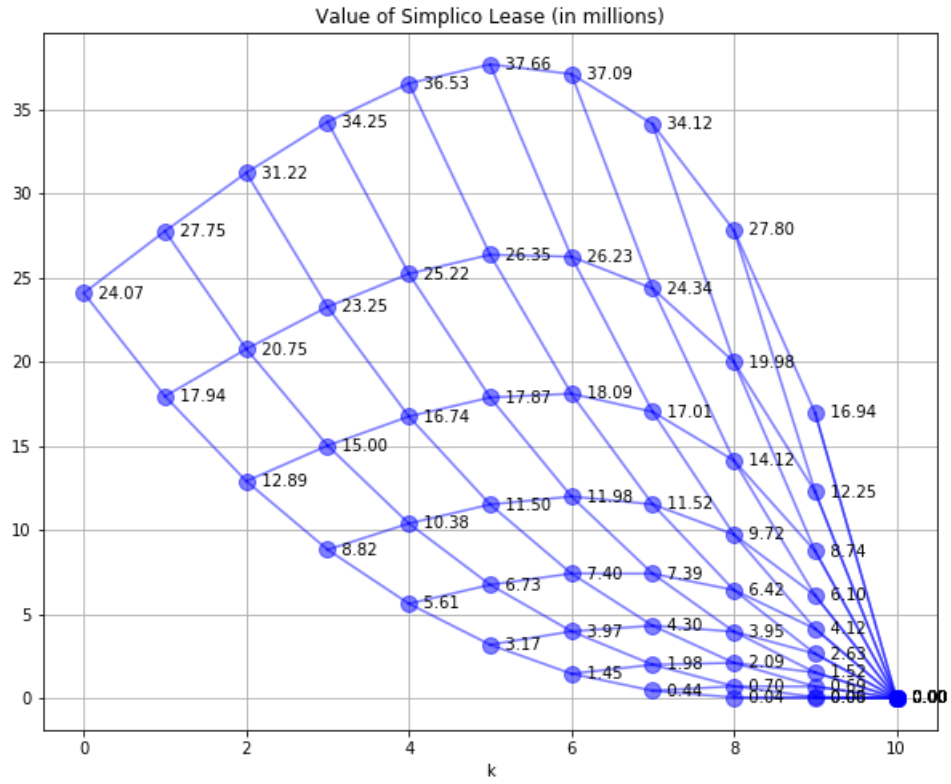
where $Q_{k,s}$ and $Q_{k,s+1}$ are the possible earnings from processing gold in stage k and state s. In this problem the lease owner has the option, but not the requirement, to process gold and may therefore elect not to operate the gold mine if the return is negative. Thus

$$Q_{k,s} = Q_{k,s+1} = \max(0, 10000(S^f_{k-1,s} - 200))$$

where is $S^f_{k-1,s}$ is the price of gold at the start of the stage and 200.

In [3]:

```
from pyomo.environ import *

# set of periods and states for each period
Stages = range(0,N+1)
States = range(0,N+1)

# create model
m = ConcreteModel()

# model variables
m.W = Var(Stages,States,domain=Reals)
m.y = Var(Stages,States,domain=Reals)
m.x = Var(Stages,States,domain=Reals)

# objective
m.OBJ = Objective(expr = m.W[0,0], sense=minimize)

# constraint list
m.cons = ConstraintList()

# definition of option value
for k in Stages:
    for s in range(0,k+1):
        m.cons.add(m.W[k,s] == m.x[k,s]*Sf[k,s] + m.y[k,s])

# value of the lease at termination is zero
for s in range(0,N+1):
    m.cons.add(m.W[N,s] >= 0)

# self-financing constraints
for k in range(1,N+1):
    for s in range(0,k):
        m.cons.add(m.x[k-1,s]*Sf[k,s] + m.y[k-1,s]*(1+r)
                   >= m.W[k,s] + 10000*max(0,(Sf[k-1,s]-200)))
        m.cons.add(m.x[k-1,s]*Sf[k,s+1] + m.y[k-1,s]*(1+r)
                   >= m.W[k,s+1] + 10000*max(0,(Sf[k-1,s]-200)))

# solve
SolverFactory('glpk').solve(m)

# post-process solution
W = {}
for k in range(0,N+1):
    for s in range(0,k+1):
        W[k,s] = m.W[k,s]()/1e6

# display
SPdisplay(W,W,'Value of Simplico Lease (in millions)')
```



The value of the lease is $24.07 million.

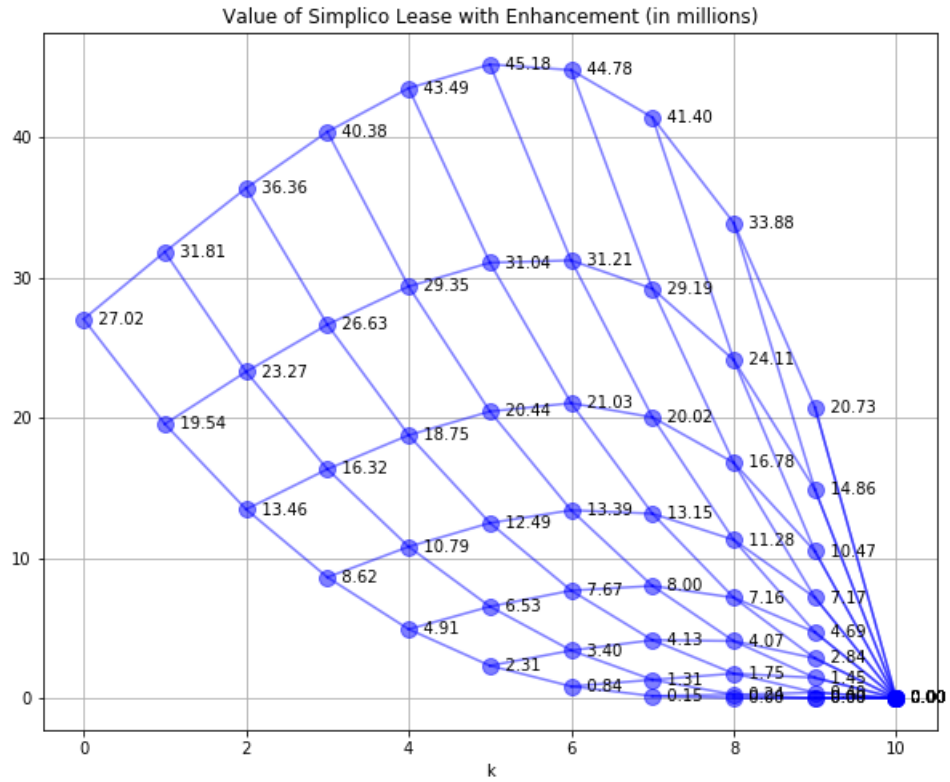
Example 3. Simplico Gold Mine Capital Investment

Next we consider an option to make a capital improvement in the gold mine following a Real Options analysis by Martin Haugh. The option increases the gold output by 25% to 12,500 oz./year. The required investment is a $4 million capital expense and an on-going increase in operating costs from $200/oz. to $240/oz. There would be no residual value resulting from the investment. The task is to determine if this option has value and, if so, how the option should be implemented.

We begin the analysis by computing the value of Simplico Gold Mine with this enhancement in place.

In [4]:

```
from pyomo.environ import *

# set of periods and states for each period
Stages = range(0,N+1)
States = range(0,N+1)

# create model
m = ConcreteModel()

# model variables
m.Woption = Var(Stages,States,domain=Reals)
m.yoption = Var(Stages,States,domain=Reals)
m.xoption = Var(Stages,States,domain=Reals)

# objective
m.OBJ = Objective(expr = m.Woption[0,0], sense=minimize)

# constraint list
m.cons = ConstraintList()

# definition of option value
for k in Stages:
    for s in range(0,k+1):
        m.cons.add(m.Woption[k,s] == m.xoption[k,s]*Sf[k,s] + m.yoption[k,s])

# value of the lease at termination is zero
for s in range(0,N+1):
    m.cons.add(m.Woption[N,s] >= 0)

# self-financing constraints
for k in range(1,N+1):
    for s in range(0,k):
        m.cons.add(m.xoption[k-1,s]*Sf[k,s] + m.yoption[k-1,s]*(1+r)
                   >= m.Woption[k,s] + 12500*max(0,(Sf[k-1,s]-240)))
        m.cons.add(m.xoption[k-1,s]*Sf[k,s+1] + m.yoption[k-1,s]*(1+r)
                   >= m.Woption[k,s+1] + 12500*max(0,(Sf[k-1,s]-240)))

# solve
SolverFactory('glpk').solve(m)

# post-process solution
Woption = {}
for k in range(0,N+1):
    for s in range(0,k+1):
        Woption[k,s] = m.Woption[k,s]()/1e6

# display
SPdisplay(Woption,Woption,'Value of Simplico Lease with Enhancement (in millions)')
```

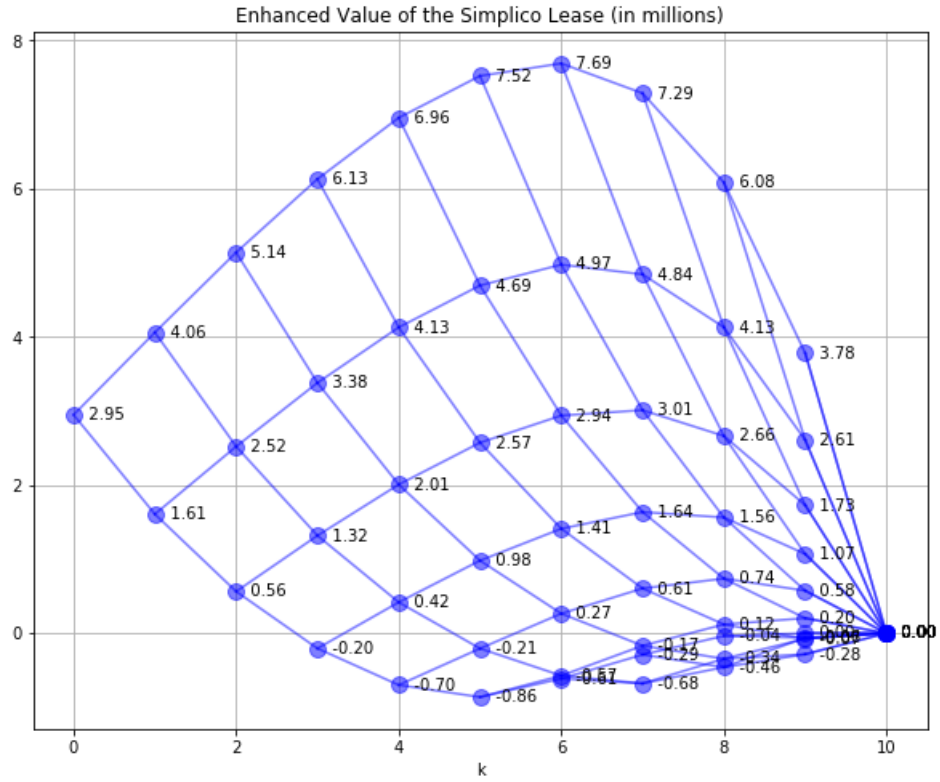


Immediate implementation of the capital expansion would raise the initial of the lease from $24.07 to $27.02 million accounting for the increase in capacity and operating cost. The increase of $2.95 million is not enough to justify an initial capital investment of $4 million.

We can see this difference more clearly by plotting the difference in lease values between the enhanced gold mine and the gold mine without enhancement.

In [5]:

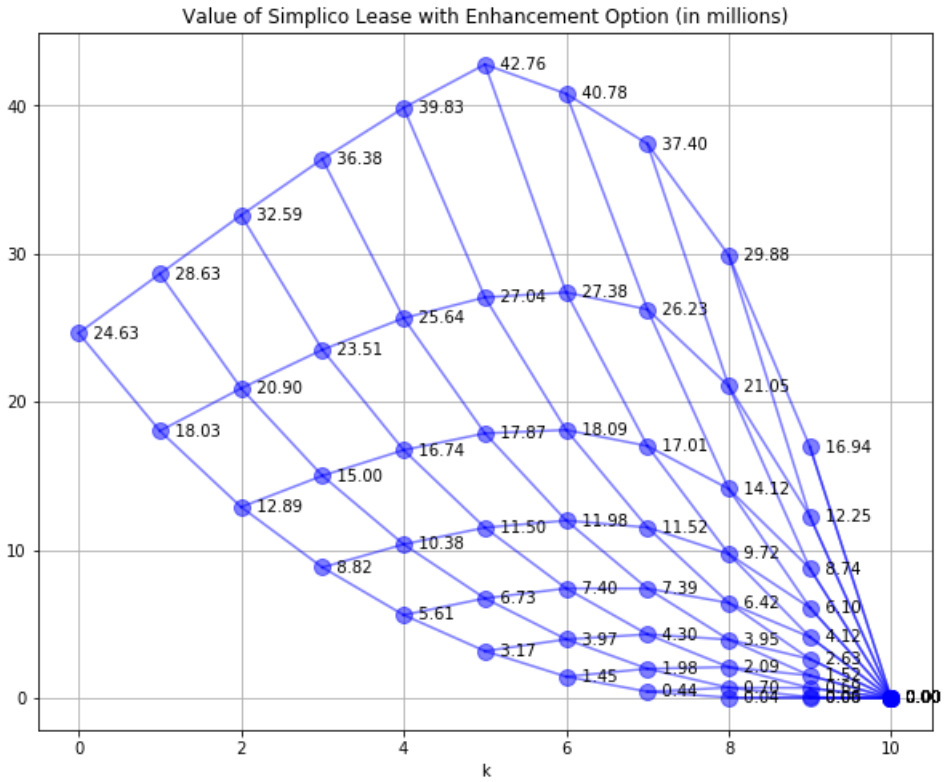
```
# post-process solution
E = {}
for k in range(0,N+1):
    for s in range(0,k+1):
        E[k,s] = Woption[k,s] - W[k,s]

# display
SPdisplay(E,E,'Enhanced Value of the Simplico Lease (in millions)')
```

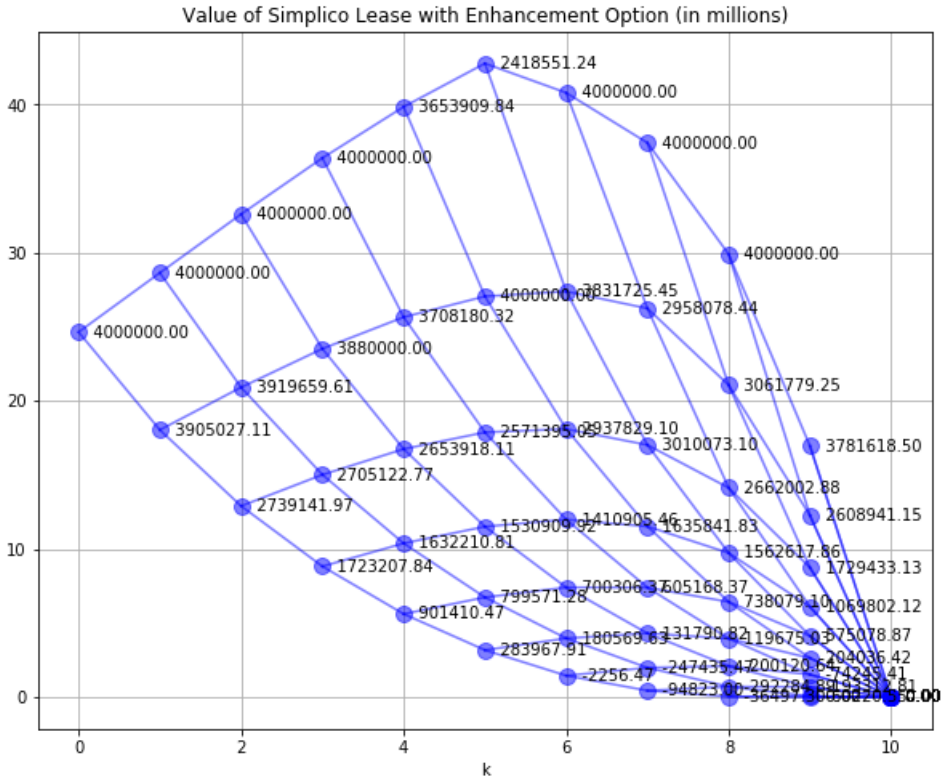


Nodes above the $4 million threshold correspond to scenarios where the operator would increase value by electing to implement the mine enhancements. Those scenarios are characterized by high gold prices occuring relatively early in the course of the lease.

In [6]:

```
from pyomo.environ import *

# set of periods and states for each period
Stages = range(0,N+1)
States = range(0,N+1)

# create model
m = ConcreteModel()

# model variables
m.W = Var(Stages,States,domain=Reals)
m.y = Var(Stages,States,domain=Reals)
m.x = Var(Stages,States,domain=Reals)
m.Woption = Var(Stages,States,domain=Reals)
m.yoption = Var(Stages,States,domain=Reals)
m.xoption = Var(Stages,States,domain=Reals)

# objective
m.OBJ = Objective(expr = m.W[0,0], sense=minimize)

# constraint list
m.cons = ConstraintList()

# definition of option value
for k in Stages:
    for s in range(0,k+1):
        m.cons.add(m.W[k,s] == m.x[k,s]*Sf[k,s] + m.y[k,s])
        m.cons.add(m.Woption[k,s] == m.xoption[k,s]*Sf[k,s] + m.yoption[k,s])
        m.cons.add(m.W[k,s] >= m.Woption[k,s] - 4000000)


# value of the lease at termination is zero
for s in range(0,N+1):
    m.cons.add(m.Woption[N,s] >= 0)
    m.cons.add(m.W[N,s] >= 0)

# self-financing constraints
for k in range(1,N+1):
    for s in range(0,k):

        m.cons.add(m.x[k-1,s]*Sf[k,s] + m.y[k-1,s]*(1+r)
                   >= m.W[k,s] + 10000*max(0,(Sf[k-1,s]-200)))
        m.cons.add(m.x[k-1,s]*Sf[k,s+1] + m.y[k-1,s]*(1+r)
                   >= m.W[k,s+1] + 10000*max(0,(Sf[k-1,s]-200)))

        m.cons.add(m.xoption[k-1,s]*Sf[k,s] + m.yoption[k-1,s]*(1+r)
                   >= m.Woption[k,s] + 12500*max(0,(Sf[k-1,s]-240)))
        m.cons.add(m.xoption[k-1,s]*Sf[k,s+1] + m.yoption[k-1,s]*(1+r)
                   >= m.Woption[k,s+1] + 12500*max(0,(Sf[k-1,s]-240)))

# solve
SolverFactory('glpk').solve(m)

# post-process solution
W = {}
E = {}
for k in range(0,N+1):
    for s in range(0,k+1):
        W[k,s] = m.W[k,s]()/1e6
        E[k,s] = m.Woption[k,s]() - m.W[k,s]()

# display
SPdisplay(W,W,'Value of Simplico Lease with Enhancement Option (in millions)')
SPdisplay(W,E,'Value of Simplico Lease with Enhancement Option (in millions)')
```





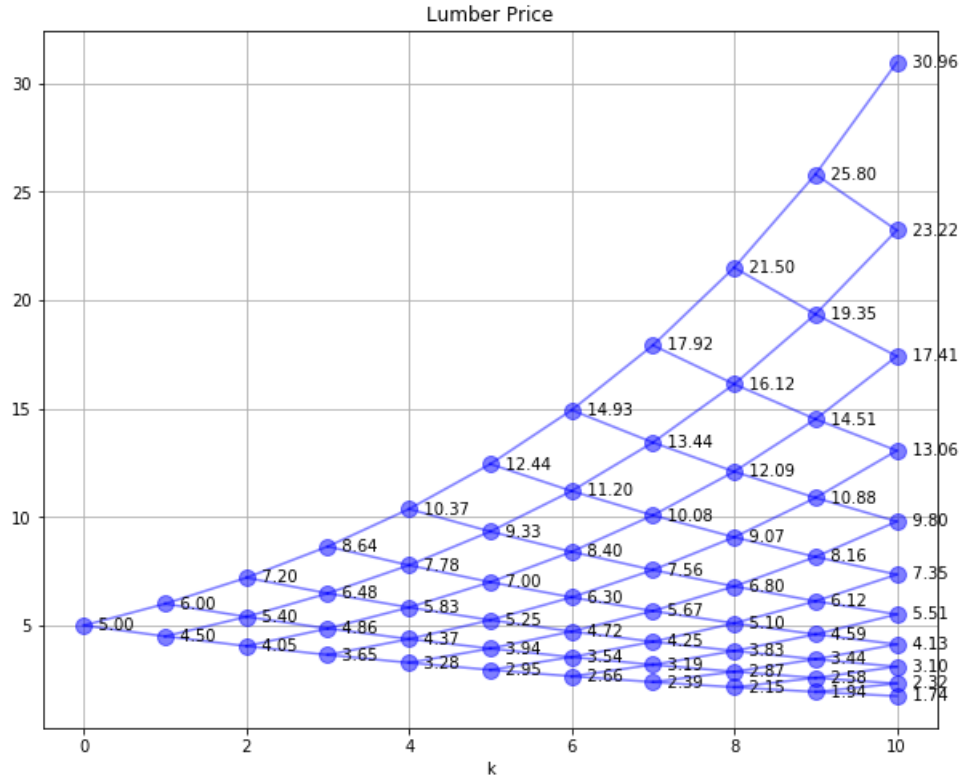
In this case the capital expense has been incorporated into the leave value calculations. Because the value of the lease has increased from the base case of $24.07 to $24.63 million, the availability of the option adds value to the lease.

Example 4. Tree Farm Harvesting

Price Model

In [7]:

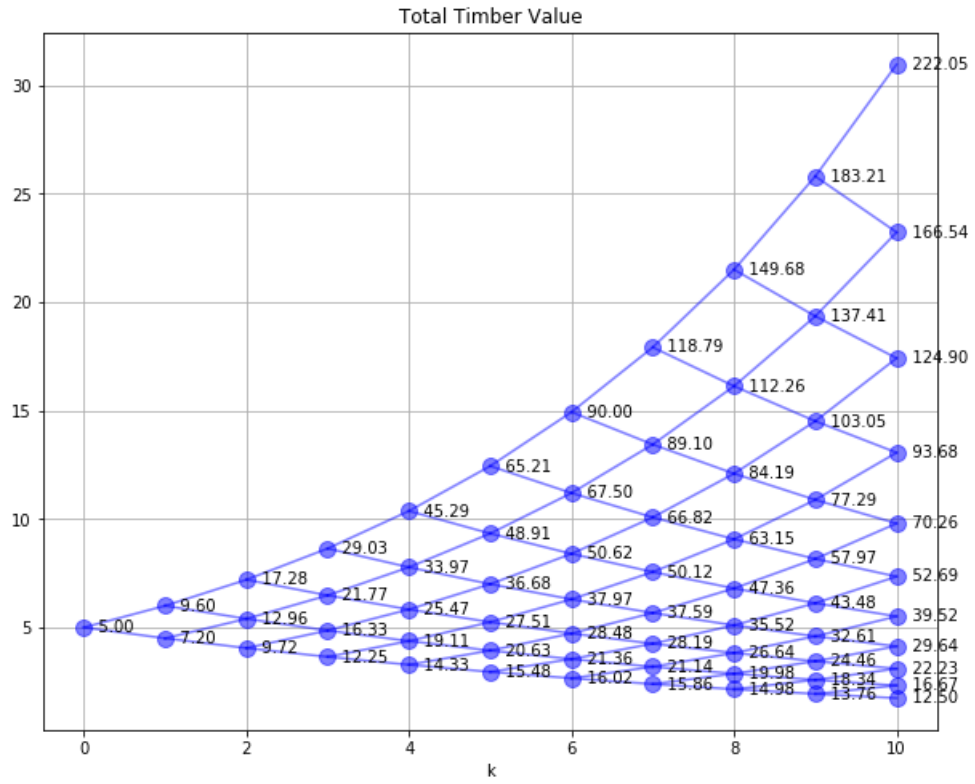
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from pyomo.environ import *

# model parameters
N = 10
u = 1.2
d = 0.9
r = 0.10

# initialize Sf
Sf = {}
Sf[0,0] = 5
for k in range(1,N+1):
    for s in range(0,k+1):
        Sf[k,s] = u**(k-s)*d**s*Sf[0,0]

# utility function to plot binomial tree
def SPdisplay(Sf,D,title=''):
    """
    SPdisplay(Sf,D,title)
        Sf: binomial tree of future prices
        D: binomial tree of data to display for each node
        title: plot title
    """
    plt.figure(figsize=(10,8))
    nPeriods = max([k for k,s in Sf.keys()]) + 1
    for k,s in Sf.keys():
        plt.plot(k,Sf[k,s],'.',ms=20,color='b',alpha=0.5)
        if (k > 0) & (s < k):
            plt.plot([k-1,k],[Sf[k-1,s],Sf[k,s]],'b',alpha=.5)
            plt.plot([k-1,k],[Sf[k-1,s],Sf[k,s+1]],'b',alpha=.5)
    for k,s in D.keys():
        plt.text(k,Sf[k,s],'   {0:.2f}'.format(D[k,s]),ha='left',va='center')
    plt.xlabel('k')
    plt.grid()
    plt.title(title)

SPdisplay(Sf,Sf,'Lumber Price')
```



Growth Model

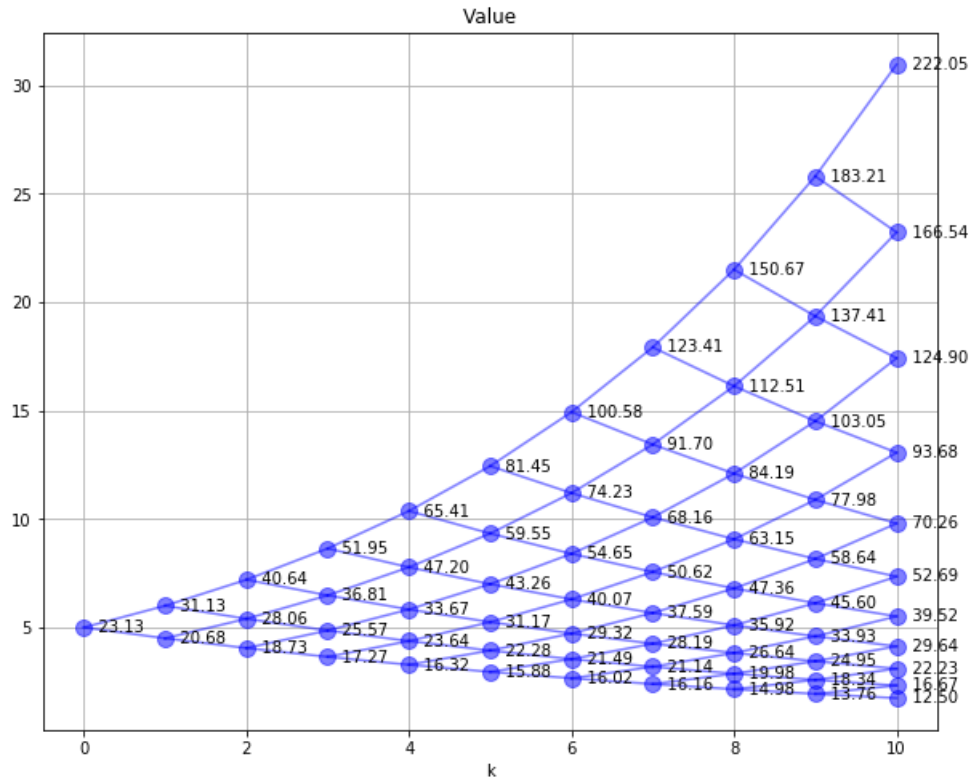
In [9]:

```
Q = {}
Q[0] = 1
Q[1] = Q[0]*1.6
Q[2] = Q[1]*1.5
Q[3] = Q[2]*1.4
Q[4] = Q[3]*1.3
Q[5] = Q[4]*1.2
Q[6] = Q[5]*1.15
Q[7] = Q[6]*1.1
Q[8] = Q[7]*1.05
Q[9] = Q[8]*1.02
Q[10] = Q[9]*1.01

QSf = {}
Periods = Q.keys()
for k in Periods:
    for s in range(0,k+1):
        QSf[k,s] = Q[k]*Sf[k,s]

SPdisplay(Sf,QSf,'Total Timber Value')
```



In [10]:

```
from pyomo.environ import *

# set of periods and states for each period
Stages = range(0,N+1)
States = range(0,N+1)

# create model
m = ConcreteModel()

# model variables
m.W = Var(Stages,States,domain=Reals)
m.x = Var(Stages,States,domain=Reals)
m.y = Var(Stages,States,domain=Reals)

# objective
m.OBJ = Objective(expr = m.W[0,0], sense=minimize)

# constraint list
m.cons = ConstraintList()

# definition of W[k,s]
for k in Stages:
    for s in range(0,k+1):
        m.cons.add(m.W[k,s] == m.x[k,s]*Sf[k,s] + m.y[k,s])

# self-financing constraints
for k in range(0,N):
    for s in range(0,k+1):
        m.cons.add(m.x[k,s]*Sf[k+1,s] + (m.y[k,s] + 2)*(1+r) >= m.W[k+1,s])
        m.cons.add(m.x[k,s]*Sf[k+1,s+1] + (m.y[k,s] + 2)*(1+r) >= m.W[k+1,s+1])

# harvest option
for k in Stages:
    for s in range(0,k+1):
        m.cons.add(m.W[k,s] >= Q[k]*Sf[k,s])

# solve
SolverFactory('glpk').solve(m)

# post-process solution
W = {}
for k in range(0,N+1):
    for s in range(0,k+1):
        W[k,s] = m.W[k,s]()

# display
SPdisplay(Sf,W,'Value')
```


Value
k
0
2
4
6
8
10
5
10
15
20
25
30
222.05
183.21
166.54
150.67
137.41
124.90
123.41
112.51
103.05
93.68
100.58
91.70
84.19
77.98
70.26
81.45
74.23
68.16
63.15
58.64
52.69
65.41
59.55
54.65
50.62
47.36
45.60
39.52
51.95
47.20
43.26
40.07
37.59
35.92
33.93
29.64
40.64
36.81
33.67
31.17
29.32
28.19
26.64
24.95
22.23
31.13
28.06
25.57
23.64
22.28
21.49
21.14
19.98
18.34
16.67
23.13
20.68
18.73
17.27
16.32
15.88
16.02
16.16
14.98
13.76
12.50

In [ ]: